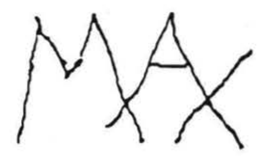


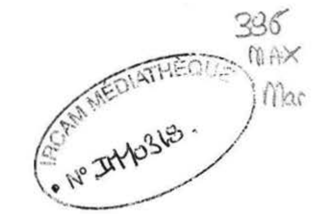

# LE MANUEL D'UTILISATION

Philippe MANOURY

Documentation
interne

novembre 1988

IRCAM
Paris



# PREFACE

L'environnement de programmation graphique de musique informatique MAX est conçu pour répondre au besoin général d'un contrôle raffiné des synthétiseurs pour l'exécution "live". Ce besoin est spécialement urgent à l'IRCAM où une partie importante de la recherche est de trouver des moyens de réaliser des compositions *electronics live* dans lesquelles les instrumentistes peuvent avoir un contrôle de plus haut niveau que ce qu'offrent les outils du commerce. A la suite de deux productions que j'avais dirigées, je pensais à la possibilité de rendre plus facile un tel travail dans l'avenir. Tandis que je commençais les premières expériences avec un système de boîtes et de lignes qui est devenu MAX , Philippe Manoury posait les bases d'une pièce pour piano et synthétiseur. Philippe n'est pas connu pour être effrayé par la technologie, même avant qu'elle ne soit totalement stabilisée, et commençait des expériences sur les chaînes de Markov, avec les prémisses de MAX , à un moment où je n'aurais encouragé personne a perdre son temps avec un programme tellement embryonnaire. Pendant que je développais MAX , Philippe commençait à l'utiliser dans sa pièce.

C'est ainsi que le développement de MAX suivait en grande partie les besoins de ce qui est maintenant appelé *Pluton*, pour piano et 4x. Le temps limité accordé à cette production nécessitait pourtant les impératifs suivants: fiabilité, facilité d'édition, rapidité d'exécution. Le risque d'un tel contexte est d'écrire un programme trop lié au specificités que requièrent une composition. J'espère y avoir résisté.

Après avoir fini la pièce, Philippe m'a suggéré d'écrire un manuel pour MAX, ce qui m'a plu. La fréquentation de toutes les versions successives du programme que je lui ai infligées, l'a obligé à acquérir une connaisance très profonde de MAX. Il a écrit ce manuel d'un point de vue musical. Ce programme a, de toutes manières, été conçu pour des musiciens comme lui, plus que pour des chercheurs comme moi.

Miller Puckette.

## INTRODUCTION ET AVERTISSEMENT

L'essentiel, pour nous artistes, est de simuler. Dans cette perspective, les machines peuvent paraitre plus sensibles. Elles peuvent suivre un jeu instrumental et réagir de manière extrêmement raffinée par rapport à ce jeu. La musique électroacoustique subit actuellement de grands développement. L'un des plus importants, à mon sens, est celui qui permet de faire en sorte qu'elle puisse être interprétée au même titre que n'importe quelle musique. Les conditions de cette interprétation introduisent en nouveau rapport entre l'instrumentiste et la musique. L'interprétation peut parfois être décisive quant au déroulement et à la nature intrisèque de cette musique. Celle-ci peut être figée dans ses constituants, comme c'est le cas avec la musique électroacoustique traditionnelle (sur bande magnétique par exemple), mais peut aussi être modelable en temps réel par l'instrumentiste. J'ai tenté d'en donner des définitions dans un petit traité intitulé: **les partitions virtuelles.** Ce fascicule expose une série de cas de figures ou la notion de partition est relativisée. A savoir que certains des composants musicaux sont fixés à l'avance, tandis que d'autres sont fonction de l'interprétation du moment. Il faut, dans ce cas, trouver les bonnes correspondances entre ce que produit l'instrumentiste et ce que produit le synthétiseur. Je considère le traité en question comme la partie théorique de ce manuel qui s'attache surtout à décrire les mécanismes qui rendent possible ces configurations.

J'ai tenté de réaliser le manuel que j'aurais aimé trouver si j'étais dans la situation d'un musicien désirant se familiariser avec un programme d'informatique musicale. C'est à dire que les exemples musicaux seront assez nombreux. Par rapport à ceux-ci, je serai toujours en accord avec ceux qui y trouveront un niveau musical insuffisant. Je leur dirai seulement que, n'ayant pas voulu entrer dans une situation proche du synthétiseur (MAX est un environnement qui n'est pas spécifiquement attaché à un système de synthèse et de traitement précis, mais peut s'adapter à diverses machines) j'ai taché d'en montrer les lignes générales et les contrôles qu'il permet. Il n'y aura donc pas de considérations sur la qualité sonore des événements qu'il produit. Il est clair que, pour nous, la composition ne peut se concevoir sans cette qualité. Imaginant que la situation la plus probable, dans laquelle se trouveront mes lecteurs, sera celle d'un Macintosh relié à un petit système MIDI (DX 7 par exemple), je n'ai pas voulu entrer dans les spécifications propres à ce mode de synthèse. J'incite donc ceux-ci à considérer ce manuel comme une exploration de diverses configurations très générales au niveau des contrôles pouvant s'appliquer à des synthétiseurs virtuels. J'en ai fait moi-même l'expérience en utilisant ce programme pour composer "*Pluton*" pour piano et le système de synthèse et de traitement en temps réel 4X fabriqué à l'IRCAM. Il en sera fait souvent allusion. Enfin, j'espère avoir été suffisamment clair et concis et ai tâché de ne pas ennuyer les lecteurs avec des considérations inutiles.

**REMERCIEMENTS**

Je tiens à remercier Miller Puckette pour l'aide qu'il m'a apporté dans l'écriture de ce manuel, ainsi que Marc Battier pour la rédaction du chapitre sur MIDI ainsi que pour la mise en forme finale.

**Philippe Manoury.**

# 1. GENERALITES

## 1.1. En quoi consiste MAX ?

MAX est un environnement qui permet de construire des configurations musicales à l'aide d'une symbolique principalement axée sur le graphique. Vous y manipulerez des boites, des potentiomètres, vous enverrez des messages à des objets qui exécuteront des actions, vous testerez des valeurs pour des paramètres et les recueillerez ensuite dans le corps de votre programme etc... bref, vous créerez un environnement qui sera adapté à vos besoins pour un cas de figure précis.

Ce programme est l'oeuvre de Miller Puckette qui l'a développé à l'IRCAM en 1987-88. Conjointement à l'écriture de ce programme, j'ai composé "*Pluton*" pour piano et 4X qui l'utilisait pour établir la communication entre les deux. L'écriture de la partition et celle du programme ont donc été contemporaines, chacune bénéficiant de l'autre pour de nouvelles possibilités. La validité musicale d'un tel travail a donc été prouvée.

La principale idée de MAX est celle de *virtualité*. En effet, ce programme n'est pas adapté à un synthétiseur précis, mais fournit des patchs virtuels pour gérer des valeurs dans ce synthétiseur.Un patch est une connexion de différents modules devant communiquer entre eux. Imaginez une série de machines très conventionnelles comme les composants d'une chaîne HiFi, vous avez les hauts-parleurs, l'amplificateur, le lecteur de cassettes, la radio etc... Toutes ces machines ont leur fonctions propres et sont reliées entre elles par des câbles. Ainsi, si vous voulez enregistrer la radio à partir de votre lecteur de cassettes, il vous faudra placer un cordon qui reliera la sortie de la première à l'entrée de la seconde pour faire traiter l'information. MAX est conçu pour cela, le résultat d'un calcul que vous avez effectué quelque part sera dirigé au moyen d'une connexion vers un module quelconque qui lui même sera redirigé vers un autre module etc... cela constitue un patch. Un patch est une connexion qui peut servir à un moment précis de votre composition mais qui peut être inutile dans une autre configuration, il faut le changer ou le modifier suivant les besoins locaux de votre travail. Ainsi votre programme se présentera comme une succession de patchs dans le temps comme si vous aviez changé les connexions en cours de route. Ces patchs ensuite changeront automatiquement sans aides extérieures. Le type d'informations qui transitera dans MAX sera pour une grande partie du code MIDI. MIDI est une norme internationale permettant de coder différents aspects de la musique, les plus usuels, dirons-nous, comme les hauteurs, intensités, débuts et fins de notes etc...MAX recevra donc des données MIDI, les traitera puis retournera d'autres données MIDI issues des traitements spécifiques qu'on leur fera subir. Le cas de figure classique que vous verrez dans ce manuel sera souvent celui d'un synthétiseur à clavier MIDI (DX7 par exemple) fournissant, lorsque vous jouerez, des informations à MAX, qui, à son tour, déclenchera des actions dans le synthétiseur. Mais avant d'aborder le programme proprement dit, voyons en quoi consiste la norme MIDI.

## 1.2 Le standard MIDI

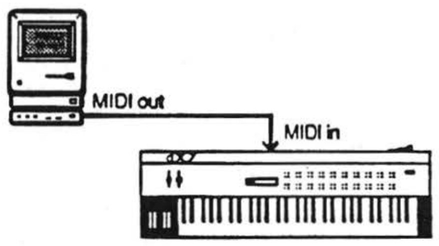
### 1.2.1. Introduction

Le standard MIDI (Musical Instrument Digital Interface) définit un cadre de communication entre des dispositifs numériques. Ce standard est né du besoin d'interconnecter des synthétiseurs de fabrication variées, et a été développé par des constructeurs américains au cours de l'année 1982. Depuis son apparition, son application s'est étendue en direction de tous les appareils numériques à vocation musicale: synthétiseurs (comme un DX-7, un échantillonneur ou un processeur numérique de son), ordinateurs (équipés d'une interface MIDI), claviers de contrôle, appareils de transformation du son (réverbérateur, délais), et même instruments de musique pourvus d'un moyen de transformation de son jeu en données MIDI.

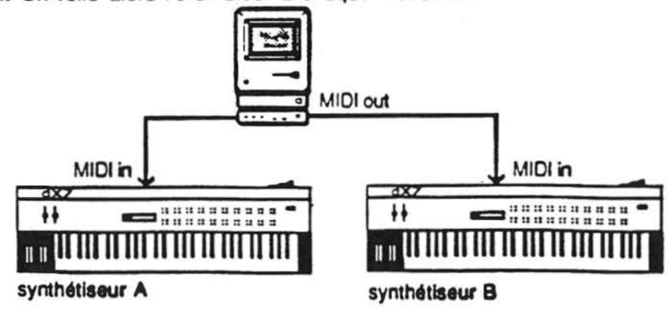
Dans la manipulation de MIDI, vous devez distinguer entre l'organe *émetteur* de données, et le *récepteur*. Dans de nombreux cas, un instrument peut combiner ces deux états, et est capable de produire des données MIDI et d'en recevoir: il en est ainsi pour un synthétiseur numérique. Par contrôle, il existe des dispositifs chargés d'engendrer des données MIDI, sans avoir besoin d'en recevoir. Un tel système est appelé un *contrôleur* MIDI. Ce peut être un clavier muet, permettant de transmettre des informations MIDI de différentes natures (il est souvent nommé un clavier-maître, ou master keyboard); dans une catégorie différente, un instrument de musique conventionnel possédant au surplus la capacité de produire des données MIDI en échange de son jeu est aussi un contrôleur: il en existe aujourd'hui plusieurs, dont la flûte, le piano ou le vibraphone.

### 1.2.2. Un réseau de communication musicale

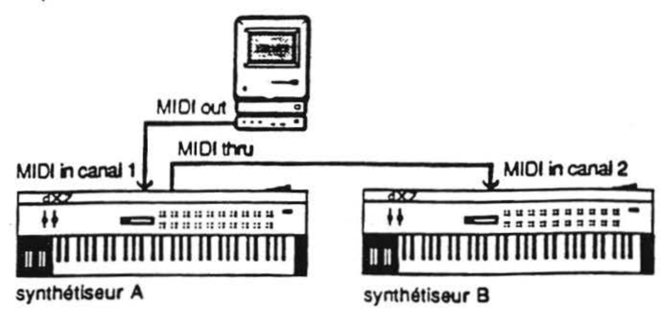
Prenons l'exemple d'un ordinateur qui exécute un programme de partition musicale, et qui transmet, via MIDI, les notes à un synthétiseur. La communication s'établit dans un seul sens: l'ordinateur, équipé de son interface MIDI (l'organe **source**), et le synthétiseur (la **destination**).




Dans un second cas, le même ordinateur contrôle plusieurs synthétiseurs: chacun jouera la même partition. On relie alors l'ordinateur à chaque instrument.



Si l'on souhaite que chaque synthétiseur joue une partie différente, il faudra employer un moyen offert par MIDI: les *canaux*. Chaque instrument sera réglé pour recevoir sa partie sur un canal déterminé, différent de l'autre. Le synthétiseur A recevra le canal 1, tandis que le synthétiseur B recevra le canal 2. Un câble relie la prise **MIDI Out** de l'interface de l'ordinateur aux prises **MIDI In** des synthétiseurs. Le standard MIDI permet aussi au canal 2 de traverser le synthétiseur A sans le faire jouer: une prise MIDI particulière sera utilisée, pour véhiculer les données MIDI de l'ordinateur vers le synthétiseur B, à travers le synthétiseur A: la prise**MIDI Thru**.




Lorsque un plus grand nombre de dispositifs MIDI est réuni, il faut avoir recours à des appareils spéciaux qui permettent l'interconnection de leurs canaux MIDI.

Le "MIDI Patch Bay", ou dispositif de brassage MIDI, reçoit les signaux provenant des sources, et les distribue vers n'importe quelle destination. Ce principe permet de réaliser une interconnection physique entre différents appareils, sans avoir à toucher aux câbles; il permet aussi d'envoyer le flux MIDI d'une source sur plusieurs destinations.

La fonction "MIDI Merge" (fusion) doit tenir compte de la nature numérique des signaux MIDI: elle est réalisée par le dispositif de brassage MIDI, ou bien par un boîtier indépendant. Elle permet de fusionner plusieurs canaux MIDI en un seul.

### 1.2.3. Structure des signaux MIDI

Le standard MIDI repose sur le principe de canaux, qui conduisent l'information musicale d'un organe-source vers une destination. Il offre 16 canaux, numérotés le plus souvent de 1 à 16 (attention, vous pouvez rencontrer des systèmes qui spécifient les canaux de 0 à 15). Le canal véhicule des données de différentes natures, qui reflètent certaines catégories du jeu musical.

| *Informations musicales* | *Données MIDI* |
|---|---|
| les codes de hauteur | pitch-MIDI |
| les codes d'intensité | vélocité |
| les code de pression | after-touch |
| les code d'instrument | program change |
| les codes de portamento | pitch bend |

Il n'est certainement pas nécessaire de connaître le système binaire pour employer MIDI: le programme MAX offre de nombreux objets qui manipulent l'information musicale et la convertissent au format MIDI. Toutefois, voyons la résolution de la représentation des données MIDI. Chaque mot d'une donnée MIDI comporte 10 bits: 1 bit "start", 1 bit de "stop", et 8 bits de données, souvent subdivisés en 2 groupes de 4 bits. Les deux bits extérieurs "start" et "stop" ne donnent aucune information musicale: ils sont là pour assurer une transmission sans erreur, et sont de plus supprimés dès la réception. On peut donc considérer qu'une information MIDI est codée sur 8 bits, soit la longueur conventionnelle d'une zone mémoire qu'on appelle plus communément un octet. Un mot de 8 bits peut représenter $2^8$ valeurs, soit

256. Pour mémoire rappelons qu'un mot de 7 bits ne représentera que la moitié, soit $2^7$ = 128. Or, le standard MIDI réserve un bit de l'octet pour sa propre gestion; il s'agit de ce que appelle le *bit de poids fort*, ou MSB (*Most significant bit*). Ce bit sert à indiquer si l'information véhiculée est contenue sur un ou plusieurs octets: le plus souvent, il faudra plus qu'un seul octet pour la représenter. S'il on ôte ce bit, qui sert à indiquer que le message MIDI est contenu sur plusieurs octets qui se suivent, on constate que le standard MIDI ne permet de coder que des valeurs sur 7 bits, de 0 à 127.

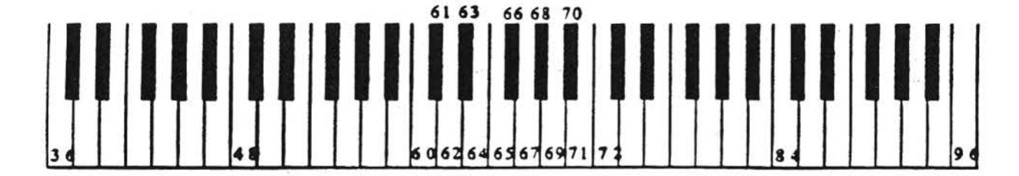
Ainsi, les hauteurs seront représentées sur 128 valeurs. Si l'on souhaite coder les hauteurs chromatiques, cela est amplement suffisant: un ambitus de 128 notes couvre notre étendue de la perception des hauteurs usuelles, si l'on se réfère à l'univers instrumental. En revanche, cette résolution ne permet pas de représenter les micro-intervalles: il n'y aurait pas assez de valeurs! Il est aisé de comprendre dès lors que le standard MIDI a été conçu pour la communication de jeux de claviers, chromatiques par définition, et que c'est là une limitation importante. En réalité, ce sera à l'instrument récepteur d'interpréter les codes de hauteurs pour leur faire jouer des micro-intervalles, selon un réglage qui leur est propre.

On voit aussi un des paradoxes de MIDI: les codes MIDI de hauteurs ne représentent pas des notes précises, mais seulement la position de ces notes sur un clavier conventionnel: c'est ce qu'on appelle un *pitch-MIDI*. Ainsi, le pitch-MIDI 60 désigne la touche qui, sur un piano, donne par convention la note *do3*.



*pitch MIDI 60*

Attention, il est possible que votre synthétiseur ne donne pas la note *do3* en échange du pitch-MIDI 60, c'est qu'il est accordé différemment. En effet, les codes de MIDI de pitch se réfèrent aux touches d'un clavier, non aux hauteurs qui y sont associées. Si vous employez, par exemple, un échantillonneur, vous pouvez distribuer les sons de n'importe quel registre sur une zone quelconque du clavier. Vous pouvez aussi avoir recours à des sons inharmoniques, à hauteur indéterminée. Ce sera leur position par rapport à la distribution des notes d'un clavier qui donnera leur pitch-MIDI. Voici un guide des pitch-MIDI sur un clavier de 5 octaves. Vous trouverez à la fin de ce manuel un tableau complet des pitch-MIDI.




## 1.2.4. La transmission des données MIDI

La fréquence de transmission des données MIDI est de 31 250 Baud, soit 31 250 bits par seconde. Comme un mot MIDI complet se compose de 10 bits, la transmission s'opère à 3 125 mots MIDI par seconde. Il faut donc à chaque mot MIDI 320 µsecondes pour être transmis.

### 1.2.5. Les messages Canal

MIDI distingue plusieurs catégories de messages: elles définissent le protocole de communication selon les types de fonction à assurer, et la nature des informations. Sommairement, la catégorie des messages **Canal** regroupe les messages **Mode** et les messages **Voix** (parfois appelé aussi message *Son*). Lorsque vous travaillez avec MIDI, ce sont des informations de ces catégories que vous allez manipuler. Elles concernent la transmission des notes elles-même, la pression (*after-touch*), les contrôles de portamento (*pitch-wheel*), les changements de contrôle (*control change*), les changements de son (*program change*). Signalons qu'il existe d'autres types de commandes MIDI, qui sont regroupées dans les catégories de messages **Système**: vous en trouverez une description sommaire plus bas.

### 1.2.6. Les messages Voix

Pour chaque note transmise, MIDI a besoin d'un minimum d'informations: le *numéro du canal* sur lequel s'effectue la transmission, le *pitch-MIDI* et la *vélocité*. Cette suite de données est appelé un message **Note On**. Dès la réception du message, la note est émise. Pour l'arrêter, il faudra une commande spéciale, un Note Off. Ce message a la même structure que le Note On: le numéro du canal, le pitch-MIDI, et une vélocité quelconque. Notez que certains organes de synthèse ne reconnaissent pas le message Note Off. Dans ce cas, ils recoivent un Note On avec une vélocité nulle, qui est interprétée comme une Note Off de vélocité de valeur 64

MIDI associe la notion d'amplitude d'un son à celle de vélocité, en référence à la vitesse de frappe d'une touche de clavier. La vélocité est codée sur 128 valeurs, comme tous les codes MIDI. Vous pouvez choisir quel paramètre affecter avec cette donnée: il suffit d'associer la valeur de vélocité recueillie sur l'instrument de contrôle à un élément déterminé de l'instrument de production du son.

Les synthétiseurs numériques offrent un choix de sons possibles. On désigne ces sons sous les noms de *patches*, d'*instruments* ou *d'instruments virtuels*, ou encore des *voix*. Le standard MIDI les nomme des *programmes*. La commande de sélection des patches (**Program Change**) fait partie de la catégorie des messages Voix. Elle est associée à un numéro de canal, ce qui permet de changer un instrument associé à un canal particulier sans changer les autres (à ce sujet, voir plus bas les messages Mode).

Une série d'autres fonctions nous font pénétrer dans une dimension différente du jeu musicale: le contrôle continu. Grâce à plusieurs fonctions, MIDI permet de modifier dynamiquement des paramètres. Ce sont les commandes de pression (after-touch), de portamento (pitch-wheel), et de modulation (modulation wheel). La première est associée à la pression gestuelle exercée sur un clavier de synthétiseur pendant que la note est émise. Le clavier doit bien entendu être construit de telle sorte qu'il soit sensible à la pression. L'étendue des valeurs émises par l'after-touch correspond à la norme: de 0 à 127. Vous pouvez choisir l'effet émis par la valeur et les changements de pression sur un ou plusieurs paramètres de votre instrument. Un emploi fréquent consiste à choisir un son selon l'état de l'after-touch; en ce cas, un seuil est choisi: si la pression est inférieure, le son A est émis, sinon le son B est sélectionné. Une solution dérivée permet de mélanger les sons A et B en proportion de l'after-touch. Cette fonction permet aussi de recueillir des données *dynamiques*, c'est à dire qui varient au cours-même d'une note, et de les associer à tel ou tel paramètre de l'organe producteur de son. Deux autres procédés sont employés pour produire des contrôles continus: le portamento (*pitch-bend*, dans le jargon MIDI), associé à un dispositif gestuel sous forme d'une molette, ou **Pitch-Wheel**; l'effet de ce contrôle est souvent gradué en demi-tons, puisqu'il est associé au contrôle de hauteur du son émis. Une autre catégorie de contrôles dynamiques regroupe ce que MIDI appelle d'une manière générique les contrôleurs. Vous en voyez un exemple dans la molette appelée **Modulation Wheel** d'un synthétiseur à clavier du type Yamaha DX-7. Le standard MIDI a réservé la possibilité de recevoir jusqu'à 128 contrôleurs. En réalité, seuls 64 contrôleurs sont définis actuellement, et les valeurs de contrôleur s'étendant de 122 à 127 sont destinés à des messages Canal. Ils permettent de sélectionner les modes Omni, Poly et

Mono exposés ci-dessous, l'état du contrôle local et l'envoi d'un ordre global de Note Off (*All Notes Off*).

### 1.2.7. **Les messages Mode**

Le standard MIDI définit quatre modes de communication entre la source et la destination. Ils organisent trois catégories de transmission, qui affectent la manière dont l'émetteur et le récepteur traitent les canaux MIDI: **Omni**, **Poly** et **Mono**.

La signification de ces appellations n'est pas intuitive, surtout lorsqu'on les trouve associées. Ces fonctions sont combinées de quatre manières, que l'on appelle les **messages Mode**.

| | |
|---|---|
| Mode 1: | Omni On / Poly |
| Mode 2: | Omni On / Mono |
| Mode 3: | Omni Off / Poly |
| Mode 4: | Omni Off / Mono |

**Mode 1: Omni On/Poly**

L'instrument se règle pour recevoir les données MIDI provenant de n'importe lequel des 16 canaux. Il permet de jouer plusieurs notes à la fois. Ce mode est connu simplement sous le nom de "mode omni".

**Mode 2: Omni On/Mono**

Tous les canaux MIDI sont reçus, mais l'instrument récepteur ne joue qu'un son à la fois. Vous n'avez que fort peu de chances de rencontrer ce mode, car il n'est pratiquement jamais utilisé.

**Mode 3: Omni Off/Poly**

Les informations MIDI sont transmises sur un seul canal; l'instrument récepteur se règle sur un seul canal. C'est le mode sur lequel se règle le DX-7 à sa mise en route (mode *par défaut*). Toutefois, sur un synthétiseur multi-timbral tel que le DX-7 II qui opère une division en deux groupes de sons, le mode Omni OFF/Poly permet à chaque groupe d'être piloté par un canal MIDI différent.

**Mode 4: Omni Off/Mono**

Ce mode est particulièrement intéressant pour un synthétiseur à plusieurs timbres ou un échantillonneur. Il permet d'associer un canal MIDI déterminé à un son en particulier. Pour un échantillonneur, un canal MIDI activera un son, ou "échantillon", ou un groupe choisi de sons. Chaque canal MIDI d'une séquence à plusieurs voix pourra ainsi jouer avec un son différent.

### 1.2.8. La fonction de contrôle local

En dehors de ces informations, les messages Mode offrent des fonctions supplémentaires, dont le **contrôle local** (*Local Control*). Il est utilisé en principe pour les synthétiseurs numériques pourvus d'un clavier. Si le mode est activé (*Local Control On*), la liaison entre le clavier et l'organe de production de son (synthèse ou échantillonnage) est coupée. Le jeu du clavier est envoyé sur la borne MIDI Out, et l'organe de synthèse recoit ses commandes de l'entrée MIDI In. Ainsi, les données MIDI engendrées par le clavier peuvent être traitées par un dispositif extérieur (tel qu'un ordinateur ou un processeur MIDI spécialisé) avant de parvenir au module de production de son. Le mode par défaut est *Local Control Off*.

### 1.2.9. Les messages Système

Ils regroupent trois catégories d'informations: les messages **communs** (*Common*), les messages **temps-réel** (*Real-Time*) et les messages de **système exclusif** (*System Exclusive*). Vous aurez moins l'occasion de rencontrer ces types de données que de manipuler des messages canal.

Les messages communs affectent l'ensemble des canaux MIDI, et véhiculent des informations qui concernent la sélection d'une séquence parmi un groupe (*song select*), de synchroniser différents séquenceurs (*song position pointer*), ou de modifier l'intonation des organes de synthèse (*tune request*). Les messages de système exclusif concernent un appareil particulier. C'est par ce moyen que la norme MIDI peut traiter des cas qui échappent à la généralisation, puisque chaque organe de production ou de traitement de son possède ses particularités, ses réglages qui lui sont propres. Enfin, les messages temps-réel offrent une série de commandes qui traitent les problèmes de synchronisation entre différentes sources MIDI, tels que des séquenceurs.

### 1.2.10. Les fichiers MIDI

Lorsqu'un séquenceur pilote un synthétiseur en lui envoyant des données MIDI, il lit un fichier dans lequel toutes ces données ont auparavant été stockées. Selon les programmes de séquencement, il existe différents manières de coder ces données: un programme de notation musicale devra ajouter des informations qui échappent à la codification prévue par MIDI: articulations, modes de jeu, etc. En fait, chaque programme possède son propre format de représentation des données musicales. Ceci est une gêne considérable si l'on souhaite profiter des caractéristiques particulières de chaque programme, et combiner leurs actions. C'est pourquoi un protocole de représentation des données MIDI, au sein d'un fichier, a été mis au point.

L'auteur du format des fichiers MIDI, Dave Oppenheim a ainsi permis que des données engendrées par un programme soient reconnues et utilisées par un programme de nature et de provenance différente. MAX est capable de créer et de lire des fichiers MIDI. Ces fichiers sont également lus par des programmes commerciaux tels que Performer, Master Tracks Pro, MIDI Paint ou le séquenceur MIDIMAC 2.6 de Opcode. Des programmes convertissent les données binaires d'un fichier MIDI en *texte*, c'est-à-dire en informations lisible par l'œil humain. MAX fournit aussi un moyen de convertir les fichiers MIDI en fichier-texte. Dès lors, il est également possible de produire des fichiers engendrés par votre propre programme d'assistance compositionnelle, et de les convertir en fichiers MIDI par MAX .

## 2. POUR DEBUTER EN MAX SANS SYNTHETISEUR

Cette partie se propose d'étudier les éléments de base de MAX sans l'aide de synthétiseur. Il est, en effet, important de bien se familiariser avec l'interface graphique avant de commencer la réalisation d'exemples musicaux. Il sera ainsi possible de comprendre les principes de base *chez vous* à l'aide du seul MacIntosh même si vous ne disposez pas de synthétiseur MIDI.

### 2.1. LE MENU

Lorsque vous entrez dans **max** en *cliquant* sur **max.doc** le programme votre écran se présente ainsi: un menu et une fenêtre que nous appelerons "fenêtre de base".



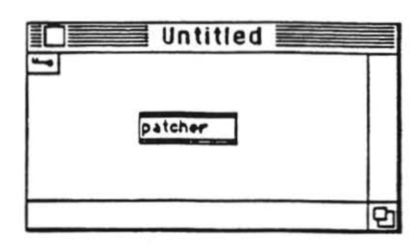

*Cliquez* sur **new**, puis amenez votre souris sur **patcher** (ou bien tapez directement **<command>p**, (la touche <command> est marquée du symbole sur les claviers standards des MacIntosh II ) vous avez les équivalences en accès rapide lorsque vous sélectionnez les différents menus). En relâchant le bouton de votre souris, une fenêtre s'affiche, comportant aussi en en-tête une série de symboles, représentés par des graphiques différents, et que nous allons voir maintenant :

1 2 3 4 5 6 7 8 9 10 11

**2.1.1** Le premier de ces symboles est une clé. Cela indique que le système comporte deux états: *verrouillé* ou *non-verrouillé*. En *cliquant* sur cette clé vous vous apercevrez que les autres symboles apparaissent et disparaissent. Le verrouillage du système (lorsque les symboles ont disparus) signifie que vous pourrez utiliser le patch que vous avez construit, mais pas en changer la configuration interne. Lorsque vous *déverrouillerez* le système (en *cliquant* à nouveau sur la clé), vous serez en position de construire votre patch, c'est à dire d'organiser les éléments le constituant et d'établir les connexions, comme on le verra plus tard, mais sans possibilités d'utilisation. Une commande rapide permet les mêmes opérations en appuyant sur la touche <command> et en *cliquant* n'importe où dans la fenêtre.

**2.1.2** Le second symbole est un **objet** dont il faudra écrire le nom dans la boîte. Un objet n'a pas de fonction propre (comme la clé par exemple) avant que son nom soit tapé à l'intérieur de cette boîte. Cette fonction et sa structure se révélera en fonction du nom. Un objet doit avoir été défini dans les sources du programme avant d'être utilisé, mais vous disposez pour l'instant d'une grande quantité d'objets standards, et nous allons étudier le plus général d'entre eux, c'est à dire l'objet **patcher** . Pour le créer, amenez votre flèche à l'intérieur du cadre le représentant et *cliquez* a l'intérieur de ce cadre mais en laissant enfoncé le bouton de votre souris. Déplacez ensuite votre souris dans la fenêtre, et relâchez la pression sur la souris, vous voyez que vous avez créé une copie du symbole sur lequel vous aviez cliqué.

Cependant vous n'avez pas créé l'objet, seulement les conditions nécessaires à sa création. Pour cela il vous suffit de taper <patcher> sur le clavier (ce qui s'imprimera automatiquement dans la boîte que vous avez sélectionnée). *Attention: si vous cliquez en dehors de l'objet, rien ne se passera lorsque vous taperez le nom de l'objet sur le clavier, il vous faudra cliquer à nouveau dans la boîte.* Un pointeur clignotant dans la boîte vous indique d'ailleurs si vous êtes en position d'écrire ou non. Déplacez la souris de manière à obtenir la flèche quelque part dans la fenêtre (mais hors de la boîte en question) et *cliquez*. A ce moment une autre fenêtre apparait identique à la première. Vous avez créer un patcher à l'intérieur d'un autre patcher.




Cette simple manipulation vous montre un concept important de ce programme, celui de ***couches superposées***. On peut, en effet concevoir des patchs contenant des patchs, et cela à n'importe quel niveau de profondeur ou vous voulez vous situer. Ainsi un patcher sera votre pièce, comportant un certain nombre de patchers représentant les différentes parties de votre pièce, qui comporteront elles-même d'autres patchers, représentant les sections, et ainsi de suite.

Remarquez quelques spécificités.

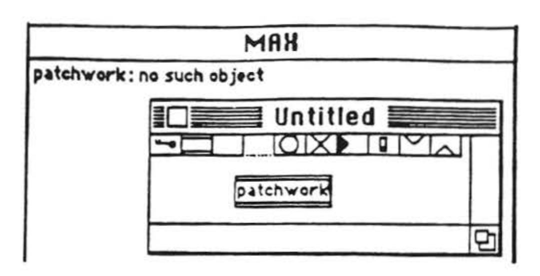
*disparition :* conformément aux autres outils MacIntosh en *cliquant* sur le carré blanc en haut à gauche vous supprimerez votre fenêtre.

*apparition* : pour faire apparaître de nouveau votre patcher, il vous faudra *verrouiller* votre fenêtre, et faire un *double-clic* dans votre objet.

*taille* : les symboles en bas à gauche vous permettent de modifier les dimension de votre fenêtre.

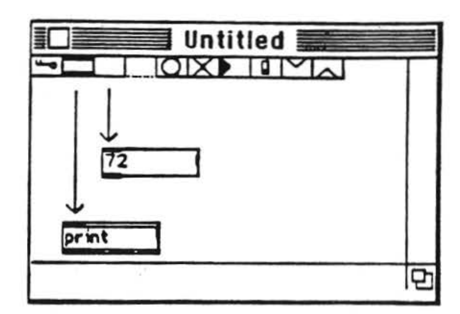
Sélectionnez **load dans** le menu **file** puis *cliquez* deux fois dans **multipatchers** (*vous pouvez sélectionner votre fichier par zônes. En effet, pour accélérer la recherche de celui-ci, vous pouvez taper la première lettre de son nom et vous serez positionné dans la zone alphabétique le concernant, ex: tapez "m" puis cherchez dans cette zône l'application* **multipatchers**) pour avoir un exemple de partition avec des patchers à plusieurs niveaux. Entraînez vous à naviguer à l'intérieur, en passant d'un patcher à l'autre, pour vous familiariser avec cette approche. Nous verrons plus tard d'autres objets de base afin de construire des patchs plus complexes.

Une remarque. Si vous tapez dans une boîte **objet** un mot inconnu, ou avec une faute, un message inscrit dans votre fenêtre de base vous signalera l'erreur comme dans l'exemple suivant.




**2.1.3** ▭ Le troisième symbole qui se présente ensuite est un **message**. Le principal rôle de cette fonction est de passer un message à un objet. Un message peut avoir diverses formes dont la plus courante est une valeur numérique. L'exemple suivant va mieux vous faire comprendre cet état de chose. Revenez dans le patcher que vous aviez créé précédemment (attention de *verrouiller* le premier patcher) en *cliquant* dans la boite. La fenêtre s'ouvre à nouveau.

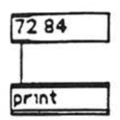
Créez un nouvel objet comme précédemment, mais au lieu de <patcher>, tapez <print>. **print** est un objet qui imprimera ce que vous voudrez sur la fenêtre de base. Une fois cet objet créé, effectuez la même manipulation avec la boite **message.**Tapez à l'intérieur une valeur numérique :




L'objectif est simple, faire imprimer la valeur numérique sur votre fenêtre de base. Vous avez créé le message, ainsi que l'objet qui recevra ce message. Il ne reste qu'à faire comprendre au système que ce message est bien destiné à l'objet en question. Pour cela, il faudra établir une connexion entre les deux boites. Vous apercevez que votre boite de message contient un carré noir en bas à gauche, alors que votre boite objet en contient deux, en haut et en bas à gauche. Il s'agit des entrées et des sorties des boites. Les entrées sont toujours en haut et les sorties en bas. Chaque objet a sa propre structure, ainsi vous vous apercevrez que l'objet **patcher** et l'objet **print**, par exemple, n'ont pas la même structure.

Pour *établir la connexion* entre vos deux boîtes, il vous faudra *cliquer* sur la sortie de votre boite **message** (tout en laissant appuyer votre doigt sur la souris). Vous vous apercevez alors que vous "tirez un fil" de cette boite. C'est le fil de raccordement que vous pouvez comparer aux cordons de votre chaîne stéréo. Vous amenez ce fil jusqu'à la boite **objet** et vous relâchez la pression du doigt sur la souris lorsque vous êtes quelque part *à l'intérieur* de la boite. Le fil se raccorde automatiquement à l'entrée. Voilà votre connexion établie.

Pour *supprimer une connexion*, sélectionnez le fil en *cliquant* dessus, puis tapez<command x>.




Pour vérifier le bon fonctionnement de ce patch,

a) *verrouillez* la fenêtre

b) *cliquez* dans la boite **message**

vous voyez alors s'imprimer sur la fenêtre de base "print: 72" à chaque *clic* que vous effectuez. SI vous changez la nature du message en imprimant par exemple <72 84>, vous verrez imprimer "print: 72 84".

Notez bien ceci : *la boîte* **message** *est un véritable éditeur de texte. Vous avez à votre disposition les principales fonctions de celui-ci comme* **cut, paste, copy** etc...

Vous savez désormais créer un message, un objet et les connecter entre eux, c'est la base de MAX.

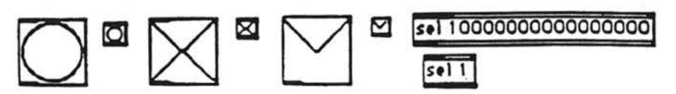
**2.1.4** Le quatrième symbole (représenté avec des pointillés dessous), est un **commentaire.** Le commentaire n'a pas d'utilité fonctionnelle, par exemple on ne peut pas le connecter avec une autre boîte, mais il a une utilité mnémotechnique. Il est, en effet, très utile, certaines fois, de mettre des commentaires dans ses programmes pour se souvenir des différentes fonctionnalités des objets que l'on utilise. Le commentaire recoit purement et simplement du texte. Créez un commentaire et tapez dedans <ceci est un commentaire>. Vous vous apercevez que la boite, trop petite pour contenir cette phrase, s'est allongée, et a coupé les mots de manière anarchique.

ceci est |
un
commenta
ire

En *cliquant* à l'extérieur de la boite, vous remarquez que celle-ci disparait, ne laissant plus qu'un petit trait sur la droite. En *cliquant* sur ce trait et déplaçant votre souris vers *la droite,* pour l'allonger, vous changerez les dimensions de votre boîte de manière à reconstituer votre phrase.

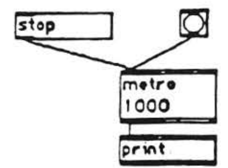
ceci est un commentaire |

**IMPORTANT:** tout les symboles graphiques de MAX peuvent être ainsi modifiés dans leur dimension, entraînez vous à cela.




**2.1.5** Le cinquième symbole est un bang. Cette fonction, très importante, sert à déclencher un processus quelconque. L'exemple ci-dessous va nous familiariser avec lui. Créez un nouveau **patcher**, avec un **objet** dans lequel vous inscrirez <metro 1000> à l'intérieur. Cela signifie que vous avez créé un métronome qui sera réglé à la seconde (l'argument 1000 = 1000 millisecondes).Connectez au métronome un autre **objet** <print>

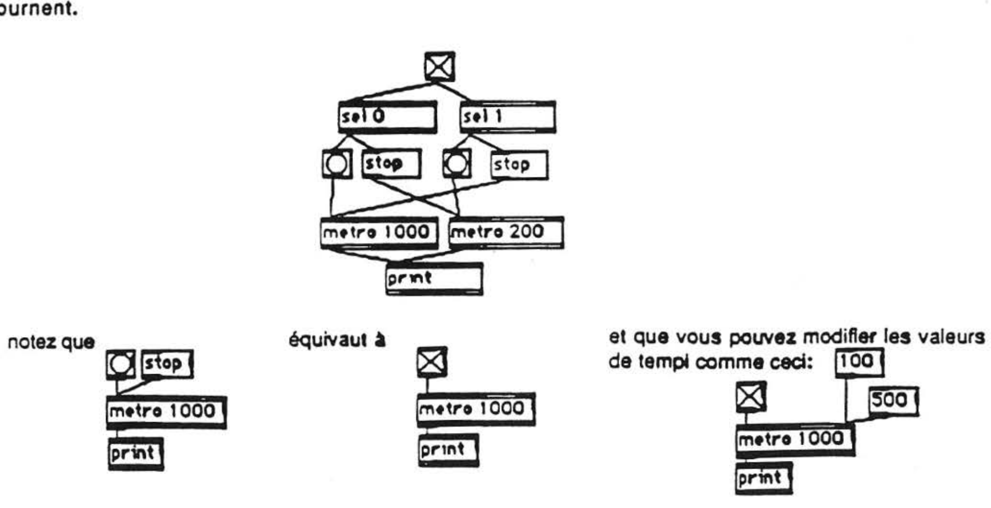
pour vérifier le résultat. Admettez que vous voulez pouvoir lancé ce processus et le stopper quand il vous plaira. Il vous suffit de connecter à l'entrée *gauche* de votre métronome deux fonctions, l'une sera un **message** <stop>, l'autre un **bang** (n'oubliez pas que les entrées des boîtes se trouvent en haut et les sorties en bas) comme ci-dessous :




Lorsque vous *verrouillerez* la fenêtre, vous *cliquerez* alternativement sur **bang** et **stop** pour lancer et stopper le processus, vous constaterez les messages "print:bang" sur votre fenêtre de base à la vitesse qui vous aviez indiquée. Pour l'arrêter, *cliquez* sur **stop.**

**2.1.6** ☒ Le sixième symbole est un **toggle**. Sa fonction est très simple. Il s'agit d'un système à deux états, qui ne sort que deux valeurs : 0 et 1. Lorsque vous *cliquez* dedans vous faites apparaître et disparaître une petite croix, ceci signifie que vous fermez et ouvrez alternativement. Sa fonction est identique au premier symbole (la clé) qui est d'ailleurs aussi un toggle. Faites l'expérience de connecter un **objet** <print> à la sortie du **toggle** et vous verrez afficher les 0 et les 1 sur votre fenêtre de base.

Dans l'exemple suivant, vous trouvez deux processus différents liés deux **objets** nouveaux: les *sélecteurs* (**sel x y z ...**). Ils sélectionnent les nombres placés en argument lorsqu'on leur envoie des valeurs numériques. Dans ce cas ils recueillent la sortie du **toggle**, et suivant la valeur ils orientent le processus sur un **metro 1000** dans le cas où elle est 0, ou sur un **metro 200**, lorsqu'elle sera à 1. Notez que des **messages stop** arrêtent l'un des processus lorsque l'autre rentre en action. Lorsqu'on a une très grande quantité de processus à gérer, cette fonction est très utile pour en neutraliser certains pendant que d'autres tournent.



**2.1.7** Le septième symbole est probablement un des plus simple. Il s'agit d'un **number** qui se contente d'afficher une valeur quelconque dans votre patch pendant qu'il tourne. Dans un exemple précédent l'objet **print** affichait les valeurs dans votre fenêtre de base, mais si vous le remplacez par un **number**, celles-ci figureront directement à l'intérieur. L'exemple suivant vous en donne une utilisation avec un **toggle** :

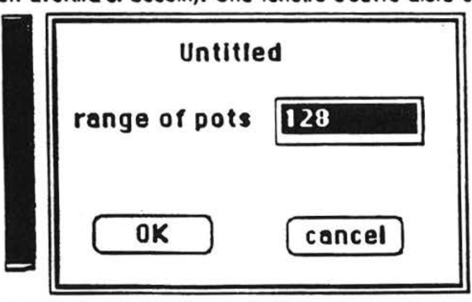



**2.1.8** Le huitième symbole est identique au précédent à ceci près qu'il est réservé à l'affichage des valeurs flottantes (telles que 1.2, 4.6 etc...)

**2.1.9** Le neuvième symbole est d'une utilisation extrêmement pratique et intuitive. Il rend d'énormes services lorsqu'on cherche à définir une valeur quelconque, c'est un **slider** ou potentiomètre. Il s'agit de la représentation graphique de ce qui serait un potentiomètre d'une console de mixage. Comme dans l'exemple ci-dessous, vous connectez la sortie d'un **slider** dans l'entrée d'un **number** et en actionnant, avec votre souris, la tirette virtuelle qui se trouve dans le milieu, vous verrez apparaître, dans le **number**, les valeurs correspondantes a la position que vous occupez. Remarquez que les valeurs extrêmes donnent 0 et 127, qui est la numérotation MIDI.



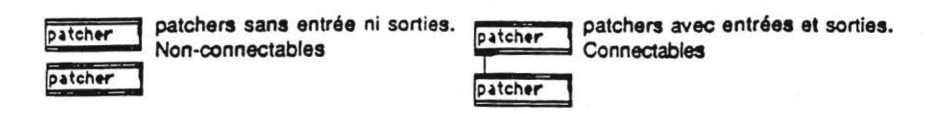

Remarquez un détail intéressant. Vous voulez changer la taille de votre slider (en hauteur) ce qui signifie que vous voulez modifier l'échelle des 128 valeurs. Sélectionnez **settings...** dans le menu **Max** après avoir sélectionner le **slider** en question (un message d'erreur "no pot selected" vous en avertira si besoin). Une fenêtre s'ouvre alors comme ci-dessous:




indiquez alors le nombre de pas voulu , puis *OK*, et vous verrez votre nouveau **slider** tel que vous le désirez. Vous pouvez aussi modifier son épaisseur comme n'importe quel autre symbole graphique.

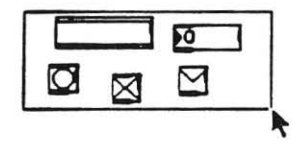
2.1.10

**Les deux derniers** symboles sont les **inlets** et les **outlets**, autrement dit, les **entrées et les sorties des** patchers. Faites l'expérience de créer un **objet patcher** à l'intérieur d'un autre **patcher**. Vous remarquerez que les contours physiques de la boîte n'ont pas d'entrées ni de sorties (les petits carrés noirs que l'on a constaté dans les différents objets). Déplacez vos deux fenêtres de manière à ce qu'elle ne se recouvrent pas et que vous puissiez les voir toute les deux. Sélectionnez un **inlet** ou un **outlet** dans votre nouveau patcher, vous verrez apparaître alors dans la boîte qui a créé ce nouvel **objet**, les petits carrés noirs qui vous indiquent que la structure de l'objet en question s'est modifiée. Ce sera la manière qui vous permettra de connecter deux objets qui ne possèdent pas d'entrées ni de sorties comme les **patchers** par exemple :




En retournant à l'application **multipatchers**, vous remarquerez que les différentes couches de patchers sont reliées à une couche supérieure par des **inlets** qui permettent d'établir les connexions entre les couches.

Pour terminer avec l'énumération des fonctions de base de MAX, voici un petit détail pratique. Lorsque vous voulez sélectionner un grand nombre de modules que vous avez créé, *cliquez* dans la fenêtre afin d'encadrer les éléments voulus (comme dans n'importe quelle application Macintosh), et vous disposez ensuite des procédés classiques (<command x> pour supprimer, <command c> et <command v> pour copier etc...).



Vous avez vu qu'il existe un hiérarchie dans les éléments que proposent MAX. Certains, tels que les **commentaires, inlets, outlets**, sont simples, d'autres sont composites. En effet, vous vous êtes aperçus que certains éléments possèdent des **outlets** (les **messages**, par exemple), d'autres, comme l'objet **metro** envoient un **bang** etc...

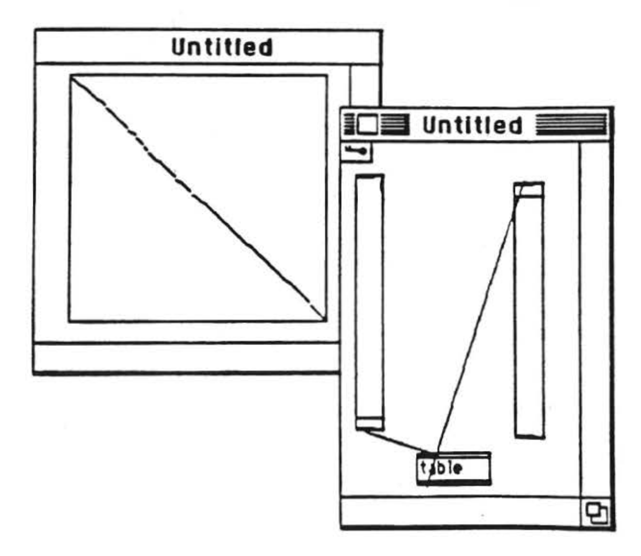
Voici un aspect intéressant qui vous aidera certainement. Il existe une documentation sur le système que vous pouvez consulter à n'importe quel moment. Lorsque vous avez un doute sur le format d'un objet, par exemple pour savoir la structure de l'objet **metro** , appuyez sur la touche Option (marquée, sur le clavier d'un Macintosh II, du symbole ⌥ ) et *cliquez* dans la boîte. Une fenêtre apparaîtra sur votre écran vous indiquant, en anglais, la signification de cet objet. Vous pouvez donc, à tout instant consulter *on line* une documentation des principaux objets usuels dont vous avez besoin. Cort Lippe, est l'auteur de cette documentation, qu'il en soit remercié.

Nous avons fait le tour des fonctions de base de MAX. Tout ce que nous utiliserons sera issu de ces fonctions (**objet, message, commentaire, bang, toggle, number, slider, inlet** et **outlet**). Certaines de ces fonctions peuvent être implémentées suivant les besoins, on peut toujours écrire de nouveaux objets, mais ceci demande une assez grande connaissance de programmation en C. Voyons maintenant quelques un des objets de base les plus usuels.

## 2.2. LES TABLES DE TRANSFERT

Nous avons vu trois types d'objet jusqu'à présent, **patcher, print sel** et **metro.** Ces objets sont définis par programmation dans les sources du programme, il est donc toujours possible de créer de nouveaux objets, mais une collection importante d'objets existe déjà qui nous permet de construire de nombreux patchs.

**table.** Il s'agit de construire des tables de transferts qui permettent de créer des connexions qui ne soient pas linéaires, ou encore de changer d'échelle. Je donne un exemple concret. Imaginez que vous voulez faire un simple *cross-fade* entre deux potentiomètres, c'est à dire que lorsque vous montez le premier, le second descend symétriquement, et vice et versa. On peut facilement résoudre ce problème par l'utilisation des tables de transferts graphiques.
Sélectionnez un objet dans lequel vous inscrivez <table>. Lorsque vous *cliquez* à l'extérieur de cet objet, vous voyez apparaître une nouvelle fenêtre de dimension plus modeste: c'est la **table.** Vous pouvez, avec la souris, tracer une courbe à l'intérieur de cette table, ce sera la *fonction de transfert.* En traçant une diagonale qui reliera le côté gauche en haut au côté droite en bas vous dessinerez la fonction de cross fade voulue. *Attention: pour obtenir un tracé très lisse, il faut dessiner très lentement.*
Sélectionnez ensuite deux **sliders** dont la sortie de l'un est reliée à l'entrée de l'objet **table**, et dont la sortie de cette dernière est connectée à l'entrée du second **slider.**




En actionnant ensuite le **slider** de gauche, vous vous apercevez que celui de droite réagit en sens contraire.

Ce processus demande une petite explication. Les tables sont basées sur les coordonnées cartésiennes et ont des valeurs stockées sur l'axe des *x* et celui des *y*. L'**objet** qui est connecté à l'entrée de la table recueillera des valeurs sur l'axe des *y*; c'est à dire que lorsque vous bougerez la tirette de votre **slider** de gauche (celui qui contrôle) dans son échelle

comprise entre 0 et 127, les valeurs correspondantes aux positions de la tirette seront recueillies sur l'axe des *y*. A chaque point de l'axe des *y* correspond un autre point sur celui des *x* qui est définit par la courbe dessinée dans la table. Le second **slider** (celui qui est contrôlé) ira donc lire la valeur de l'axe des *x* qui correspond à celle sur laquelle vous pointez dans celui des *y*. Ainsi, les valeurs élevées (127) sur l'axe des *x* correspondent, dans notre exemple aux plus basses (0) sur celui des *y* , et vice et versa.

Vous pouvez donner un nom à la table que vous venez de créer: tapez le nom à la suite, comme ceci:

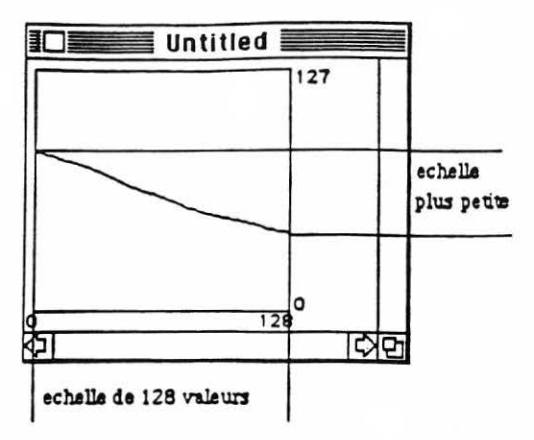
```
table tab1.t
```

Notez qu'il faut laisser un espace entre le mot "table" et le nom. *Le nom peut comporter une extension comme dans le cas précédent. Pour des raisons purement mnémotechniques, prenez l'habitude de donner systématiquement l'extension ".t" à toutes vos tables. Cela vous permettra de savoir, rien qu'au nom, la nature de certains de vos fichiers.* Lorsque votre table est dessinée, conservez-la en faisant "save": vous créez un fichier qui conservera le contenu de la table. Donnez au fichier le même nom que vous avez choisi pour la table: ainsi, dès que vous rechargerez votre patch, la table sera présente.

Faites l'expérience de modifier la courbe (*vous vous apercevrez que la réécriture d'une nouvelle courbe entraîne automatiquement l'effacement de la précédente*) et appréciez les déplacements résultants du second **slider** en fonction de celle-ci.

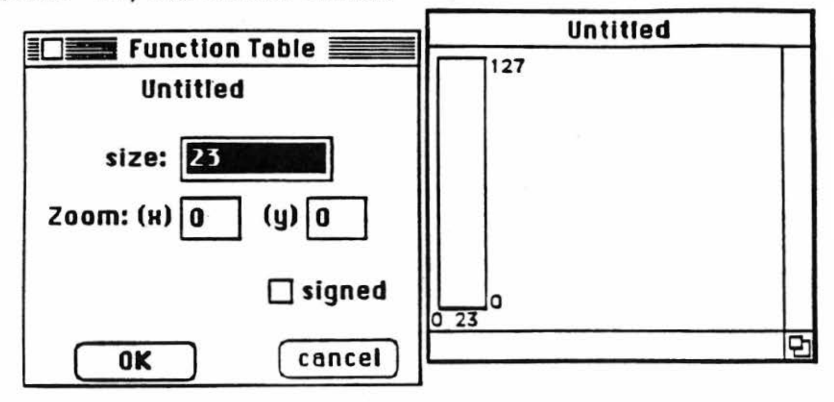
Chargez l'application **multisliders** pour voir une application multiple de ce processus, puis consultez les **tables** pour comprendre les déplacements successifs des sliders entre eux. Vous vous apercevrez que vous êtes en présence d'un système hiérarchisé en ce sens que le slider 1 qui contrôle tout les autres aura une répercussion sur le slider 6 qui sera différente si le slider actionné est un autre que le premier, en effet, chaque slider contrôlant son successeur, le mouvement sera plus complexe quand sera plus grand le nombre des sliders intermédiaires.

L'intérêt de l'utilisation de ces fonctions de transfert est énorme dans les perspectives compositionnelles. Dans ma composition *"Pluton"* pour piano et 4X toutes les relations entre les sons du piano et ceux du synthétiseur passaient par ces fonctions. Un des grands principes sur lequel je m'étais appuyé était celui de faire contrôler n'importe quel événement par la mesure de vélocité MIDI livrée par le piano. Ainsi les vélocités qui étaient étalonnées sur les 127 valeurs de l'axe des *y*, déclenchaient des sliders pouvant contrôler soit des mouvements spatiaux, soit des transpositions, soit des compressions ou dilatations d'échelles, soit n'importe quel autre phénomène voulu. Les tables permettent aussi de changer d'échelles, c'est à dire de faire correspondre une échelle de 127 valeurs à une autre plus petite comme le montre l'exemple suivant :

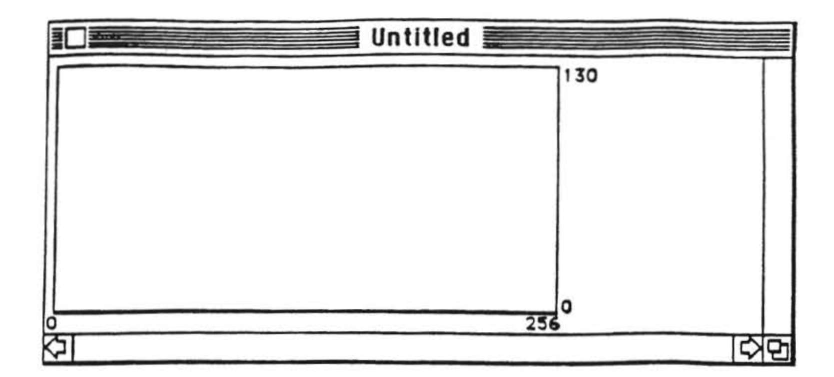



Voyons, maintenant, d'autres possibilités, plus générales, des **tables**. D'abord, vous pouvez faire varier la taille de l'axe des *y* en modifiant directement celle de votre fenêtre. L'axe des *x*, lui, ne change pas. Sélectionnez alors la fonction **settings...** dans le menu **Max** ( ou bien *tapez* directement <option w> ). Une fenêtre apparait vous offrant plusieurs possibilités.

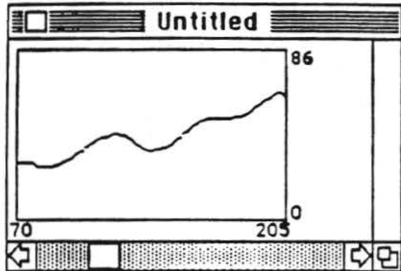
1) **size** vous demande une nouvelle taille. Si vous tapez un nombre plus petit que la norme (c'est-à-dire 128) l'axe des *x* se rétrécira,



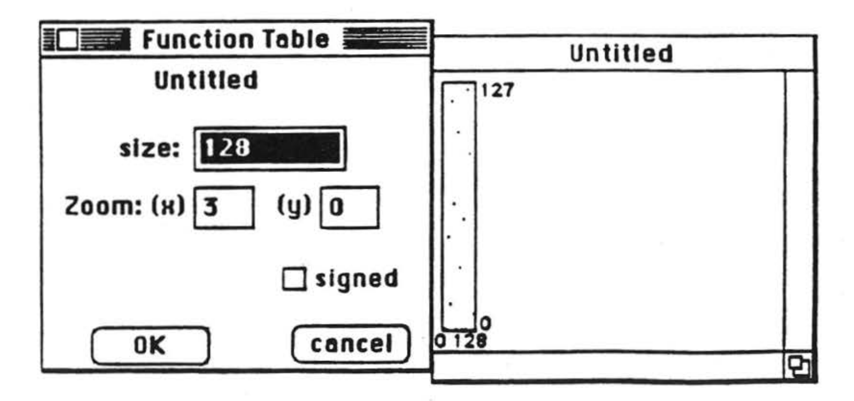

si vous tapez un nombre supérieur, il faudra élargir votre fenêtre pour avoir votre **table** dans tout sa grandeur.



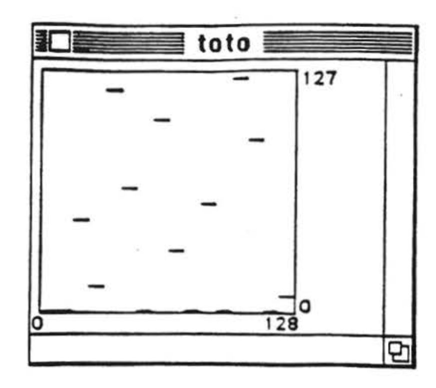

Dans le cas où la taille de votre fenêtre dépasserait celui de votre écran, un *scroll-bar* vous permet de faire défiler votre fonction:



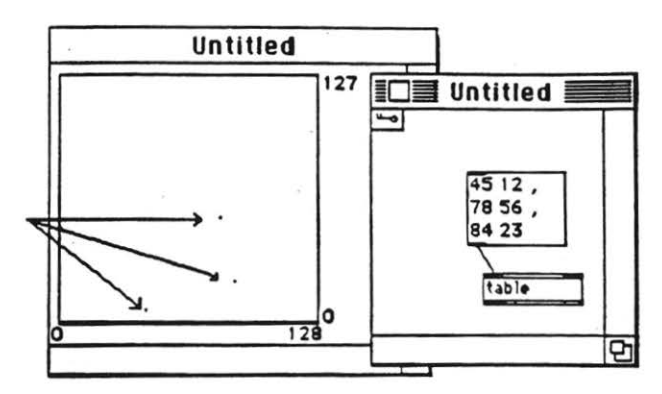

**Zoom** permet de construire des **tables** avec énormément de points. Le coefficient ( sur l'axe des *x* comme sur celui des *y* ) rétrécit la fenêtre par une puissance de 2. Ainsi lorsqu'avec un coefficient 3, comme le montre l'exemple ci-dessous, si je place des points au hasard dans la **table:**




remis à 0, le coefficient redonnera à la table sa grandeur réelle, et vous aurez alors :




Il est clair que le dessin de la courbe avec la souris reste valable dans les cas ou la configuration à traiter ne demande pas une précision *au point près* . L'application **multisliders** en est un bon exemple, il s'agit de définir des trajectoires globales dont la précision extrême n'est pas absolument nécessaire, ces cas-ci sont extrêmement fréquents. Mais on peut vouloir entrer dans la **table** des valeurs précises, ce sera le cas du suiveur de partitions. S'il s'agit de donner un petit nombre de points, la manière la plus simple est de les faire figurer dans une boîte **message** sous forme de doublets de la forme *x y* , *x y* , etc... séparés par une virgule. L'exemple suivant nous le montre :




S'il s'agit de construire une courbe plus importante, la procédure est un peu plus complexe.

1) Sélectionnez un **new** dans le menu **File** ( ou *tapez* directement <command n>). Une nouvelle fenêtre apparait qui est un éditeur de texte.

2) Tapez ensuite les valeurs de l'axe des *x* uniquement. Celui des *y* s'incrémentera automatiquement :

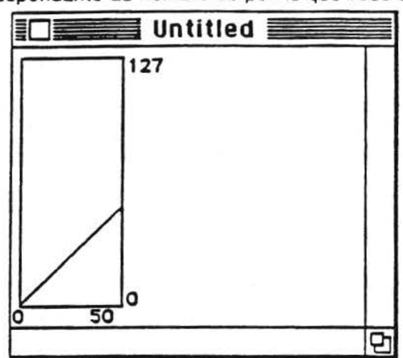


3) Sélectionnez le texte

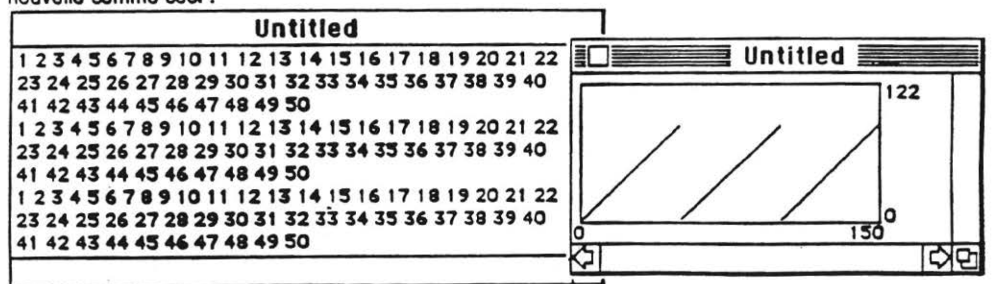



4) Faites <command c> pour le copier.

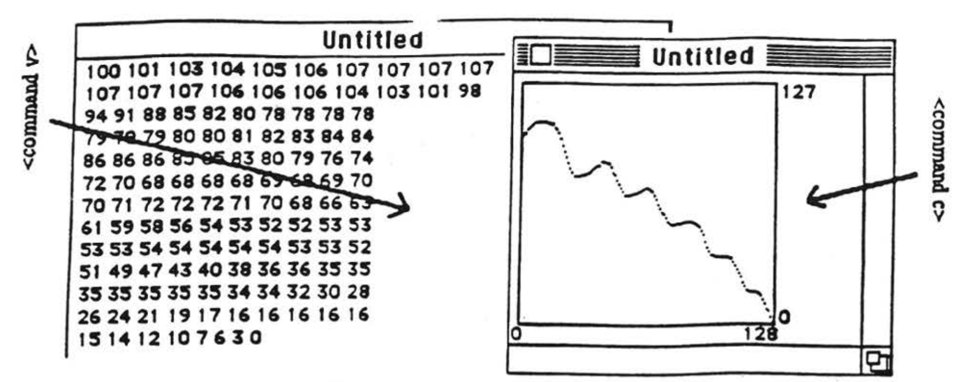
5) Enfin *tapez* <command v> quelque part dans la table, et votre courbe apparaîtra dans la dimension correspondante au nombre de points que vous avez voulu.




N'oubliez pas que vous disposez, dans cette nouvelle fenêtre, d'un éditeur de texte ou vous avez à votre disposition la plupart des outils usuels ( cut , copy, paste, save etc... ). Ainsi vous pouvez dupliquer votre courbe un certain nombre de fois de manière à en créer une nouvelle comme ceci :




Les fonctions **cut et paste** ( <command v> et <command d> ) sont aussi applicable dans l'autre sens, à savoir lorsque vous voulez faire figurer le contenu de la table dans un fichier. Dans ce cas faites <command c> dans la table et <command v> dans la fenêtre

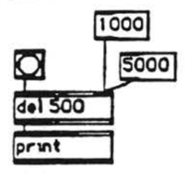



**Notez ce détail** : lorsque vous utiliserez les tables dans le contexte d'une application musicale en temps rèèl, il vous sera obligé parfois de les supprimer de votre écran, en effet leur apparition peut rallentir le système lors de l'écriture de ces tables. Ce problème ne se posera pas lors de la lecture.

## 2.3. LES OBJETS TEMPORELS

### 2.3.1 del (del x).

Ce sont des délais, c'est à dire des retards avec lesquels on peut programmer un événement. Comme pour l'objet **sel, del** est une fonction qui attends un argument qui sera la valeur du retard indiqué en millisecondes. Dans l'exemple ci-dessous, un patch très simple nous en montre une utilisation possible.
Le premier **inlet** du **del** est un **bang**. Le second permet de recevoir des messages qui changeront, si besoin est, la valeur du retard. L'exemple suivant vous montre un cas simple. Lorsque vous *cliquerez*, sur un des deux **messages** après avoir *cliqué* sur le **bang**, vous verrez s'imprimer "bang" sur votre fenêtre de base à la vitesse indiquée. Les nouvelles valeurs écraseront les précédentes dans ce cas.

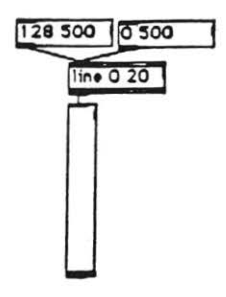



Les **del** ont énormément d'utilité a plusieurs égards. Dans *"Pluton"* il m'est souvent arrivé de figer le son du piano dans une réverbération artificielle, et pour cela il me fallait ouvrir l'entrée du réverbérateur pendant un temps assez court (entre 100 et 500 ms). C'est à dire que lorsque la note devant être réverbérée est reconnue, la valeur de l'entrée du réverbérateur doit être au maximum (c-à-d 127), et 500 ms après elle être de nouveau à 0. L'exemple **delslider** que vous allez charger simule cette situation. Le **bang** représentant la reconnaissance de la hauteur voulue enverra la valeur 127 , à votre **slider**, et le **del 500** remettra la valeur à 0, c'est à dire fermera l'entrée du réverbérateur une demi-seconde après.

### 2.3.2 line (line x y).

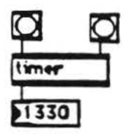
Ce processus un peu plus complexe va nous montrer un cas plus fin de l'exemple précédent. Dans l'application **delslider** les valeurs passaient brusquement de 0 à 127, sans transitions. L' **objet line** va nous permettre de réaliser ces transitions de manière continue et suivant un temps programmable. **line** est une fonction qui reçoit 3 **inlets**, 2 arguments et retourne 1 valeur. Dans l'exemple ci-dessous le premier **inlet** reçoit, au choix deux **messages** comprenant chacun deux valeurs, la première indique le pas dans le slider, et la seconde le temps en ms pour l'atteindre.

Le premier argument du **line** est un point de départ, généralement à 0 (n'y touchez pas pour le moment), le second est un grain en ms. Dans l'exemple suivant le pas changera toute les 20 millisecondes. Faites le tourner en changeant les valeurs des messages ainsi que le grain.

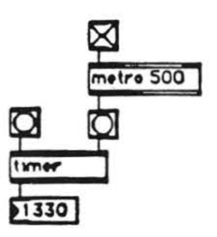



**2.3.3 timer.**

Il s'agit simplement d'un compteur qui donne le temps écoulé en millisecondes depuis un **bang**. Comme l'exemple suivant nous le montre, c'est un objet qui reçoit deux **bangs** et retourne une valeur. Celui de gauche initialise le processus à 0, et chaque fois que vous cliquez sur celui de droite, le number vous indique le temps écoulé depuis le dernier **bang** de gauche.



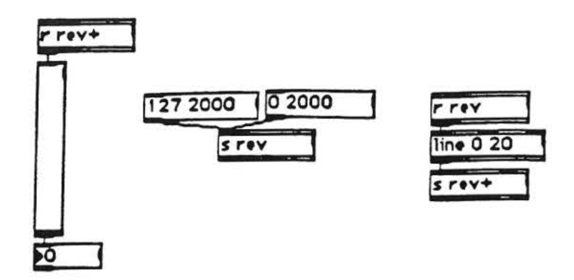

L'exemple suivant nous montre un cas ou l'affichage des valeurs se fait suivant un temps contrôlé. L'utilisation de l'objet **metro** actionnera le bang de gauche suivant le temps indiqué.




## 2.4. LES TRANSMISSIONS DE MESSAGES

**send** et **receive** (ou **s** et **r**) (**s x**, **r x**). Nous allons revoir le même exemple que **delslider**, vu précédemment, mais programmé d'une manière plus élégante. Il s'agit de l'utilisation des transmissions de messages. Ce procédé rend énormément de services, surtout en ce qui concerne la clarté et la simplicité dans les patches qui deviennent très souvent assez chargés.

Chargez l'application **sliderline**.



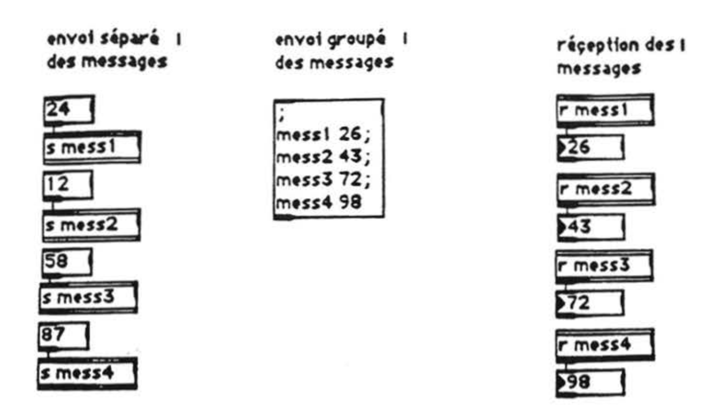

Il suffit de créer un **objet** qui sera une fonction avec 2 éléments dont le premier sera au choix **send (ou s)**, ou bien **receive** (ou **r**), et le second un nom quelconque. Ces variables **s** ou r communiqueront par l'intermédiaire de leur nom commun. Dans le cas ou elle envoie le message on utilisera un **send**, quend elle le reçoit un **receive**. Les deux **messages** <127 2000> et <0 2000> ne pointent pas directement dans le **slider**, mais pointent dans une variable intitulée **rev.** Elles chargent donc cette variable d'envoyer un message à un élément quelconque par la symbolique <s rev> tapée dans la boîte. Le programme cherche immédiatement ou se trouve la variable identique, mais, précédée d'un **r.** La variable en question passe par **le line** et pointe à son tour dans une deuxième variable (**s rev+**) qui recoit le message du **line.** Cette variable à son tour cherche le **r rev+** correspondant qui lui, pointe sur le **slider** et, de ce fait, lui transmet le message en question. Ce modèle de programmation, même s'il peut paraître plus difficile à première vue, est fortement à conseiller dans les cas ou l'on est amené à utiliser très fréquemment une fonction, comme c'est le cas avec la fonction **line.** Cela évite de la faire figurer à chaque utilisation, et l'on peut réserver un **patcher** spécialement pour cela.

L'application suivante montre une variation du même procédé mais par une lecture automatique d'une table. Il s'agit de faire parcourir à un slider tout les points d'une table. Chargez l'application **slidertable.** Vous voyez que le seul point de changement par rapport à **sliderline** est l'insertion de la table **t1.t** entre le **line** et le **send.**



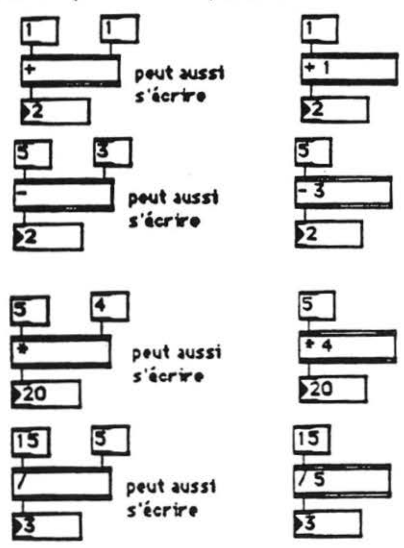

**Notez ce détail pratique:**

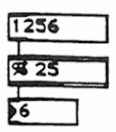
Lorsque vous voulez envoyer des valeurs au même moment à une série de **sends**, vous pouvez les spécifiez en groupe dans une même boîte **message.** Celle-ci produira un **bang général** pour tout ces **envois.** Pour cela il faut commencer par un ";", et utiliser ce ";" comme séparateur entre tout les envois. La syntaxe est : " ; <nom de la transmission> <valeur> ; <nom de la transmission> <valeur> ; etc... Cela permet d'alléger considérablement vos patches au niveau graphique. L'exemple ci-dessous vous le montre:




## 2.5. LES OPERATEURS ARITHMETIQUES

Ils sont d'un utilisation extrêmement simple. Ce sont des objets qui recoivent deux nombres (l'opérande et l'opérateur) et renvoient un argument (le résultat). Vous disposez des quatre opérateurs de base (*addition, soustraction, multiplication et division*) auxquels il faut ajouter le *modulo*. Il faut rappeler en quoi consiste le modulo. Lorsque vous divisez deux nombres *a* par *b*, le reste de la division est toujours compris entre 0 et (*b*-1). L'exemple suivant nous montre les opérateurs en question.

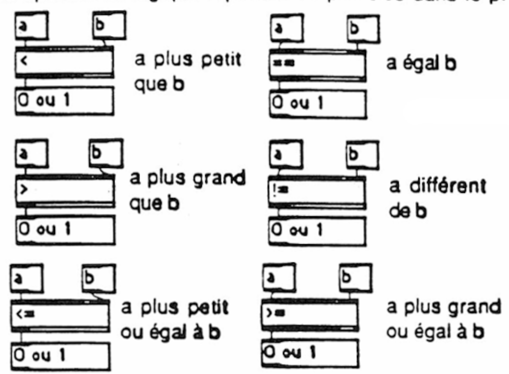





Chargez l'application **opera** pour constater l'utilisation de tout les opérateurs à partir d'un exemple déjà utilisé.

## 2.6. LES OPERATEURS LOGIQUES

Les opérateurs logiques sont des comparateurs. Ils sont chargés d'examiner des relations entre deux termes. Ces relations sont des relations logiques, c'est à dire "plus grand que", "plus petit ou égal à", "différent de" etc... Comme pour les opérateurs arithmétiques, ce sont des **objets** à deux arguments qui retournent une valeur qui, cette fois, sera binaire. Elle sera égale à 1 lorsque la comparaison est vraie, et à 0 lorsqu'elle est fausse. L'exemple suivant montre les opérateurs logiques qui sont disponibles dans le programme.

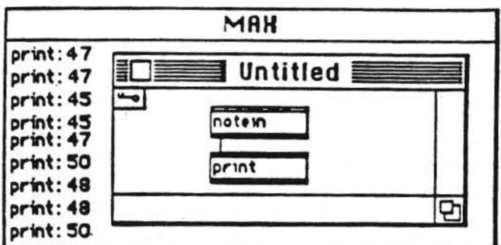



Pour avoir un exemple d'utilisation simple de ces opérateurs, lancez l'application **logic** cliquez dans le message 4000 (avant de cliquer dans le bang). Nous avons déjà rencontré un cas est identique à celui-ci, mais ici l'opérateur imprimera un 0 lorsque la valeur de temps sera inférieure à 4000, et 1 lorsqu'elle sera supérieure. Testez-le ensuite avec les autres opérateurs logiques.

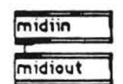
## 3. POUR DEBUTER EN MAX AVEC SYNTHETISEUR

Nous allons voir, maintenant, une série d'exemples musicaux, allant des plus simples à d'autres, plus complexes, qui vont nous permettre de raccorder ce que nous avons vu dans le chapitre précédent à une activité musicale. Nous traiterons de cas très généraux ne concernant pas des catégories de sons très précises. Donc, quelque soit le programme que vous avez dans la mémoire de votre synthétiseur MIDI, il sera suffisant pour comprendre les bases de ces fonctions de contrôles.

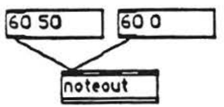
En premier lieu assurez vous de la bonne connexion de vos appareils, entre le synthétiseur, l'interface MIDI et le Macintosh. Une grande partie des problèmes proviennent souvent d'une mauvaise connexion entre les appareils. Grâce au petit test suivant, vous pourrez vérifier la validité de votre branchement. *Connectez* ces deux **objets: notein** dont la sortie gauche pointe sur **print** . En jouant sur le clavier de votre synthétiseur, vous devez voir apparaître une série de messages comme indiqués ci-dessous. Cela signifie que les valeurs MIDI sortant de votre synthétiseur rentrent bien dans le Macintosh, puisqu'il imprime ce que vous jouez.




Attention: si rien ne se passe lorsque vous jouez sur le clavier, c'est que celui-ci ne possède pas un **midi thru** (type KX88 au lieu de DX7); il faudra donc le spécifier explicitement dans notre patch par la construction suivante :




Ensuite, testez la configuration opposée. En *cliquant* alternativement sur les deux messages qui rentrent dans l'objet **noteout** vous devez entendre quelque chose sortir de vos hauts-parleurs. Dans ce cas vous envoyez des informations MIDI à votre synthétiseur qui les reçoit et les exécute en temps-réel.




Une fois passés ces tests, nous pouvons entrer dans le vif du sujet.

### 3.1. LES SPECIFICATIONS MIDI

Il existe une série d'objets permettant le codage des événements MIDI, qui chacune ont leur propre spécificité. Vous venez de voir, précédemment, une simple application de **notein** et **noteout** mais voyons un peu plus en détail comment ces objets sont construits. Il est assez important de bien avoir en tête la structure des objets courants pour pouvoir tester les

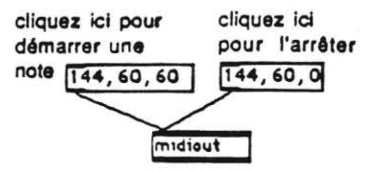
erreurs qui se commettent parfois dans les patches. *N'oubliez pas que vous pouvez la consulter "on line" à n'importe quel moment.*

### 3.1.1 midiin.

Cet objet ne reçoit pas d'arguments mais, si un message MIDI est reçu, il transmet les valeurs octet par octet à un autre objet comme le suivant par exemple. Connectez cet objet à un **print**, par exemple, et vous verrez la liste des informations MIDI apparaître sur votre fenêtre de base.

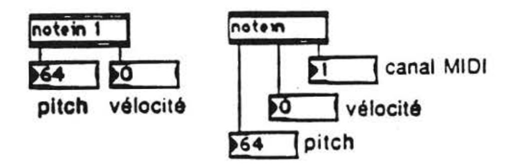
### 3.1.2. midiout.

C'est exactement le contraire du précédent, à savoir que cet objet reçoit une valeur mais n'en retourne pas. Dans l'exemple ci dessous, deux **messages** sont connectés à **midiout**, l'un pour lancer une note, l'autre pour la stopper. Testez-le, le contenu de ces message sera expliqué plus tard.




### 3.1.3. notein (ou notein n) .

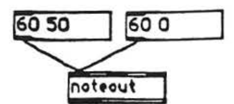
Cet objet reçoit une information MIDI du synthétiseur mais ne l'envoie pas. Plus spécifique que **midiin** il permet de recevoir indépendemment les valeurs de ***pitches***, ***vélocité*** et ***canal*** MIDI et est d'un usage plus courant. Remarquez donc, que, lorsque vous créez cet objet, il ne comporte pas d'**inlet**. Il est utile lorsque l'on veut capter une information spécifique venant du clavier. Vous avez vu, plus haut, que des messages s'imprimaient lorsque vous jouiez au clavier. Suivant le fait que **notein** prends ou non un argument, cet objet comprendra 2 ou 3 **outlets**. Comme on le voit ci-dessous, **notein 1** sort deux valeurs : le *pitch* et la *vélocité*, l'argument, lui, représente le canal MIDI alors que **notein** (sans arguments) rajoute en sortie le numéro du canal MIDI qui servait d'argument au troisième **outlet**. Remarquez que, lorsque vous jouez, vous recevez deux messages. Si vous y prêtez attention, vous verrez que, lors du relâchement de la touche, le programme vous envoie comme valeur de *vélocité* : 0. C'est le *Note Off*. Il faut savoir, que pour rejouer une même note, MIDI exige que celle-ci soit fermée. Les deux messages que **notein** vous fournit seront donc *Note On* et *Note Off* (correspondant à une valeur positive puis nulle de la *vélocité*).



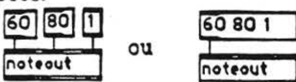

Vérifiez ce simple exemple sur le clavier en modifiant la *vélocité* (l'intensité) tout en gardant la hauteur fixe.

### 3.1.4. noteout (ou noteout n) .

Cet objet envoie une information MIDI au synthétiseur mais ne la reçoit pas de lui. C'est exactement le contraire du précédent. Comme lui, il peut avoir deux ou trois **inlets** suivant le fait qu'il possède comme argument le numéro du canal MIDI ou non. Cependant, comme vous devez lui spécifier les valeurs de manière numérique (et non directement au clavier) il est nécessaire de lui fournir le *Note Off*. C'est pourquoi, dans l'exemple :



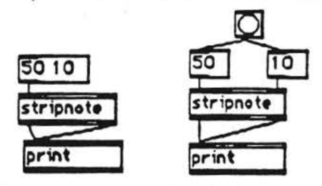

nous avions donné deux messages représentant un couple *pitch-vélocité* dont le second donnait un valeur nulle pour cette dernière. Faites l'expérience de ne *cliquer* que sur le message de gauche et vous verrez que seule la première fois sera prise en compte (cela dépend toutefois de l'état du synthétiseur, dans un mode polyphonique, les 16 premières fois seront prise en compte). Il vous faudra *cliquer* à nouveau sur celui de droite pour fournir le *Note Off*. Notez, que vous pouvez spécifier les valeurs avec un message pointant sur chaque **inlet** (correspondant donc de gauche à droite, aux *pitch, vélocités* et *canal MIDI*), ou, ce qui est plus rapide, avec une liste de valeurs tapées dans une même boîte **message** mais pointant sur l'**inlet** de gauche qui distribuera, automatiquement les valeurs aux modules *pitch, vélocité* et *canal* dans l'ordre ou elles ont été tapées.




Attention : dans l'exemple de gauche, ci-dessus, il faudra *cliquer* sur 80 ou 1 avant 60. C'est à dire qu'il faudra spécifier le *canal* et la *vélocité* (ou le contraire) avant la valeur de *pitch*. Dans l'exemple de droite cet ordre sera fait automatiquement par le programme.

### 3.1.5. stripnote.

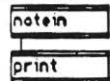
Cet objet reçoit deux valeurs numériques et les transmet séparément si la seconde est non nulle au moment ou il reçoit la première. Il possède deux **outlets** pouvant être deux **messages** comportant chacun 1 valeur numérique, ou ce qui revient au même un **message** comportant les 2 valeurs numériques. Ainsi ces deux constructions sont identiques :




Si la deuxième valeur est nulle, comme dans l'exemple suivant, vous pourrez constater que **stripnote** n'envoie aucun message à **print** :




A quoi peut bien servir un tel objet ? Imaginez que vous avez besoin d'un processus ne prenant en compte que les *Notes On*, c'est à dire le début de chaque note. C'est ce qui se passe lorsqu'on utilise un suiveur de partitions. Si vous ne disposez que de la construction suivante :

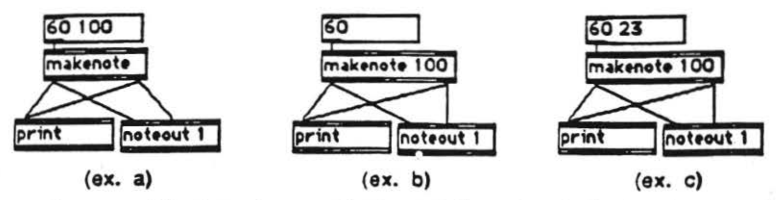



vous constatez que lors du relâchement des touches, le **print** vous envoie aussi le message pour les *Notes Off*. Alors, vous intercalez entre ces deux objets un **stripnote** qui ne vous donnera que les *Notes On*, car le deuxième argument du message qu'il recevra lors du *Note Off* sera égal à 0 puisque c'est cette valeur de vélocité qui définit justement le *Note Off* . Testez le, ainsi que le précédent:



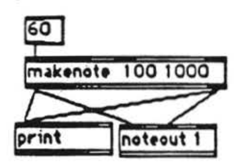

**3.1.6. makenote (ou makenote m ou makenote m n).**

Cet objet est très simple dans son fonctionnement ainsi que très pratique. Son unique fonction est d'ajouter un *Note Off* à un message pointant dans un **noteout**. Cela vous évitera donc de donner deux **messages** en entrée, comme dans l'exemple précédent. La sortie gauche de **makenote** corresponds au *pitch,* celle de droite à la *vélocité* . Ainsi , l'exemple précédent pourra être écrit, plus économiquement comme suit (ex. a) Le **print** vous permettra de vous assurez que le *Note Off* est bien créée. L'objet **makenote** peut avoir un argument qui sera une valeur de *vélocité* , dans ce cas, il n'est plus nécessaire de la signifier dans la boîte **message** (ex. b). Si une valeur différente figure, malgré tout, dans la boîte **message**, l'argument de **makenote** sera écrasé (ex. c). Faites tourner ces trois exemples et regardez attentivement les messages d'impression.



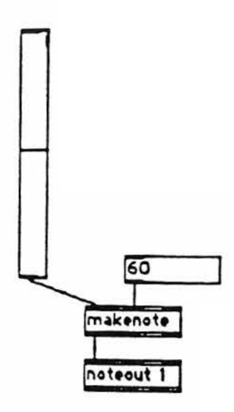

(ex. a) (ex. b) (ex. c)

Lorsqu'il en possède deux, le second argument de makenote indique le temps (en ms) après lequel la *Note Off* doit être créée. Dans l'exemple suivant, vous verrez que l'impression de la *vélocité* 0 se fait avec une seconde après l'envoi de la valeur 60 à l'objet.



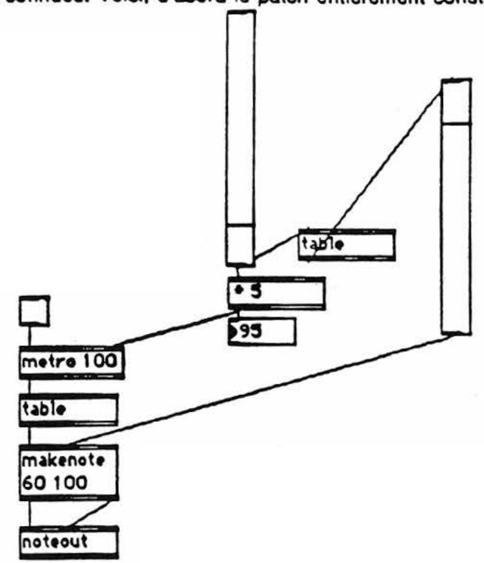

**EXERCICE**

Le petit patch ci-dessous, se propose de faire sortir les *pitches* d'un **slider**, alors que l'indice de *vélocité* est constant. En le faisant tourner, vous constatez que rien ne sort de votre synthétiseur. Tâchez de comprendre pourquoi, et complétez-le.

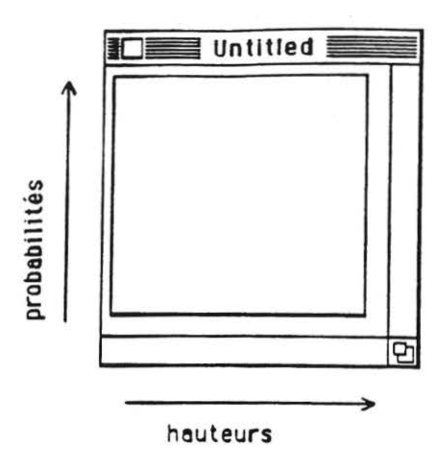



## 3.2. EXEMPLE COMMENTE

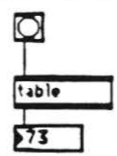
Voyons maintenant un exemple, certes limité du point de vue strictement musical, mais qui nous permettra de réaliser un patch un peu plus complexe en même temps que retrouver diverses fonctions déjà connues. Voici, d'abord le patch entièrement constitué:



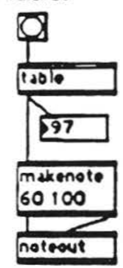

L'idée est d'avoir une succession aléatoire de hauteurs dont on puisse faire contrôler la vélocité par le tempo, tout en ordonnant ces hauteurs facilement des plus probables aux moins probables. Le moyen, le plus simple, mais peut-être pas le plus raffiné au niveau du contrôle des hauteurs, est de disposer ces variables (probabilités et hauteurs) sur une **table**, ou , sur l'axe des *x*, seraient alignées les hauteurs, alors que les probabilités le seraient sur celui des y, comme nous le montre le schéma ci-dessous.



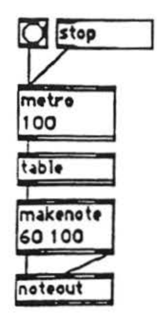

Lorsque l'on connecte un **bang** à l'entrée d'une **table**, le programme comprends cette attribution des axes dans la table, sans qu'il soit nécessaire de lui fournir d'autres explications. Donc la simple disposition suivante suffit:



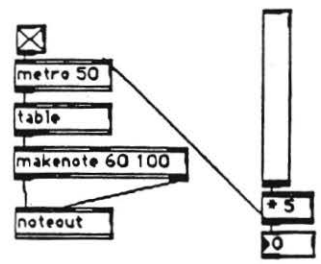

Nous allons construire, petit à petit, notre patch. Maintenant, si vous rentrez un simple point, quelque part dans la **table**, en *cliquant* dans le **bang**, la valeur de votre point s'affichera dans le **number**. Ajouter un autre point plus haut, sur l'axe des *y*, et répétez l'opération. Vous constatez que la deuxième valeur apparait plus souvent que la première. La **table** fournit les *pitches*, l'**objet makenote** fournira la *vélocité* et la *durée*, ainsi que les *Note Off*. L'**objet noteout** enverra les valeurs au synthétiseur (n'oubliez pas que le synthétiseur attends un *Note Off* pour pouvoir redémarrer, c'est la fonction de **makenote**). Ainsi la patch ci-dessous enverra les valeurs de la **table** au synthétiseur. Testez-le, et faites varier les valeurs à l'intérieur de la **table**.




Pour le réglage du tempo, il suffira, d'intercaler un objet **metro** (avec son argument en ms) entre le **bang** et la **table** sans oublier le message **stop** pour l'arrêter.

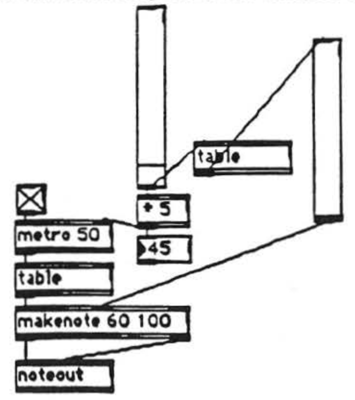



Les modifications de tempi sont ensuite apportées par un **slider** dont les valeurs sont multipliées par un coefficient pour donner les temps de départ de chaque note et dirigées vers l'entrée gauche du métronome. Notez que le **bang** et le **stop** sont remplacés par un **toggle**, et cherchez la raison de cette équivalence. Faites tourner votre patch tout en modifiant en *temps réel* le contenu de votre **table**.




Reste, pour terminer, le contrôle de la vélocité par le tempo. Un nouveau **slider** dont les valeurs entreront dans l'entrée du milieu de **makenote** modifiera les valeurs de vélocité. Une **table** située entre les deux **sliders** suffira à les faire se contrôler l'un par l'autre (cf. l'application **multisliders** vue plus haut). Choisissez un son assez chargé en spectre (type cloche par exemple) pour bien entendre les variations d'amplitude, et faites tourner cet exemple en variant le contenu des deux **tables**.
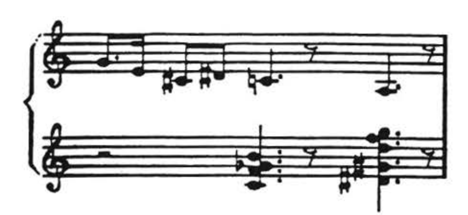
**Remarque importante**: avant de faire tourner ces exemples, vérifiez bien dans le menu  que le volume du son du Macintosh est à 0, et que le cache mémoire n'est pas actif (en position "non"). Ces deux conditions sont valables pour le bon fonctionnement de MAX.



**Attention :** n'oubliez pas de remplir le contenu de vos tables sans quoi rien ne sortira de votre synthétiseur.

**EXERCICE**

Soit le petit fragment suivant à réaliser. Vous jouez la partie supérieure sur votre clavier, le synthétiseur vous donne la partie inférieure :



Quels sont les problèmes à traiter ?

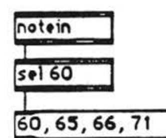
1) Le programme doit reconnaître les deux dernières notes (*do* et le *la*) car c'est à ces moments là qu'il doit exécuter les accords. Pour la reconnaissance et la sélection des valeurs numériques, vous savez qu'il existe un objet pratique : le **sélecteur**, qui aura comme argument la valeur de la note à reconnaître en pitch MIDI. Ainsi pour le *do* la construction suivante fera l'affaire :

sel 60

2) Le **sélecteur** doit recevoir les valeurs provenant du clavier pour les sélectionner. Il faut alors lui connecter un **notein** comme ci dessous :




3) Il faut envoyer les valeurs des accords au synthétiseur. Pour ce faire, créez des boîtes **messages contenant ces données**, qui seront espacées par une virgule, comme vous montre le codage du **premier accord** :




Les virgules indiquent au programme que les valeurs alignées ne font pas partie de la même liste et donc ne comprendra pas la succession 60,65 comme valeurs de *pitch* et de *vélocité* mais comme valeur de *pitch* uniquement (le premier dans la liste).
Récapitulons bien. Le **notein** reçoit les valeurs du synthétiseur lorsque vous jouez. Il les transmet au **sel 60**, qui enverra un **bang** à la liste de pitches MIDI correspondant aux notes de l'accords.

Comment procéder pour terminer ce patch ?

1) Codez d'abord la sélection du *la* final de la partition, puis la deuxième liste des valeurs MIDI du deuxième accord.

2) Construisez l'assemblage nécessaire afin d'envoyer ces valeurs au synthétiseur par l'intermédiaire **des objets** que nous venons de voir dans ce chapitre.

3) Ayez à l'esprit qu'il n'est pas nécessaire de dupliquer deux fois le même objet. Ainsi le **notein** que vous avez créé, servira pour les deux actions.

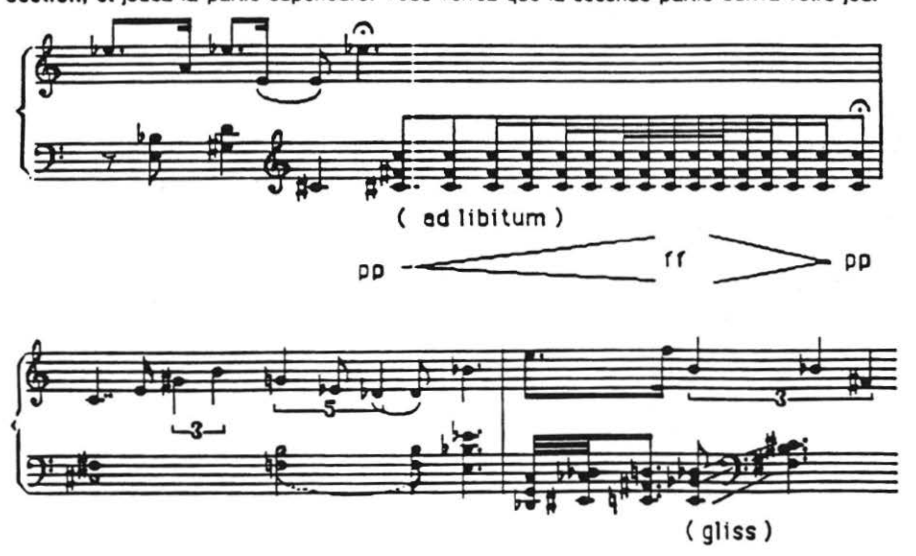
4) Une fois terminé, jouez la partition, et vérifiez que tout fonctionne parfaitement. Attention, j'ai volontairement omis de signaler un problème qui se produira si vous n'y avez pas prêter attention lors de la construction de ce patch. La sélection se produit normalement, l'envoi des valeurs des accords au synthétiseur aussi, cependant à certains moment celui-ci effectuera une action que vous n'avez pas voulue. Cherchez la cause de cette action, et tachez d'introduire un **objet** qui pourra la supprimer.
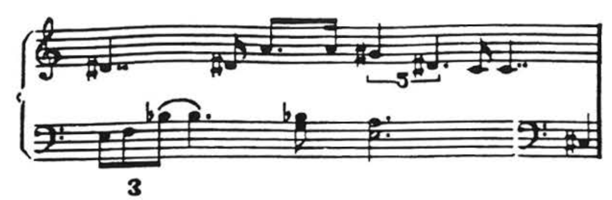
# 4. LE SUIVEUR DE PARTITIONS

On le sait, MAX est un programme essentiellement conçu pour les applications en *temps-réel.* C'est à dire que le synthétiseur doit produire des actions à un moment qui n'est pas déterminé avec précision, par rapport au début d'une pièce, mais doit pouvoir répondre *instantanément* à une action extérieure qu'il ne connait pas à priori. Cette action peut être de plusieurs ordres; cela peut être un simple bouton (on l'a vu en utilisant les **bangs** ou les **toggles**), mais cela devient vraiment intéressant lorsque ces actions sont les éléments d'une partition que le système doit reconnaître et qui doivent le faire réagir. La principale idée forte de cet état de fait est que l'instrumentiste, se trouvant réintégré dans le corps de la musique électroacoustique, est maître du jeu en ce sens que le système doit pouvoir suivre les variations de tempi qui sont inhérentes à toute interprétation. MAX comprend, outre les éléments nécessaires au contrôle et à la production des sons, un suiveur de partitions. Celui-ci aura la charge de synchroniser les éléments prévus dans le corps du programme avec la partition jouée au clavier. J'emploie le terme "clavier" pour plus de commodité dans le contexte pédagogique dans lequel je me place, mais il est clair que tout instrument produisant du code MIDI rentre dans ce cas de figure.

## 4.1. UN EXEMPLE DE FONCTIONNEMENT.

Avant d'explorer les différents problèmes liés au suiveur de partitions, voici un petit exemple. Chargez l'application **partition**, *cliquez* dans le message **1** pointant dans **s section**, et jouez la partie supérieure. Vous verrez que la seconde partie suivra votre jeu.
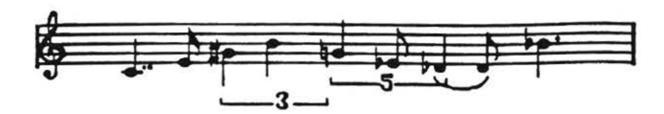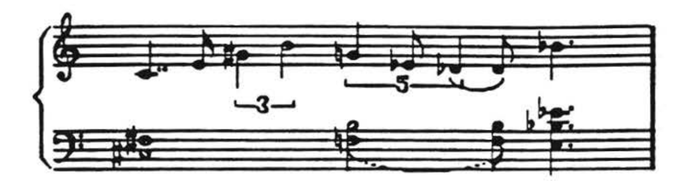





C'est cet exemple que nous allons reconstruire, pas à pas, tout au long de ce chapitre. Voici, pour commencer, un cas simple ou l'on peut faire suivre *en temps réel*, le synthétiseur par le jeu du clavier.

## 4.2. UN CAS D'UTILISATION SIMPLE.

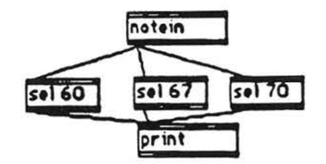
Imaginons que sur ce petit fragment mélodique :



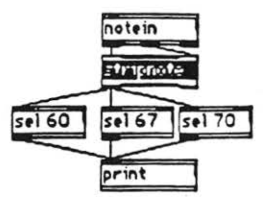

vous vouliez plaquer ce simple accompagnement de trois accords :



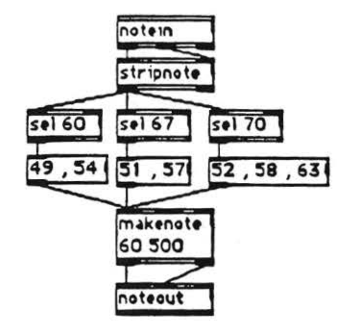

La méthode la plus simple reviendrait à reconnaître les notes devant déclencher ces accords (successivement ***do, sol bécarre*** et ***si bémol***, soit en *pitches* MIDI : 60, 67 et 70). Vous avez déjà rencontré des **objets** permettant de faire une sélection sur des valeurs numériques qu'il reçoivent : ce sont les **sel x** (ou x = la valeur numérique). Il faut donc créer trois **objets** permettant la reconnaissance des valeurs 60, 67 et 70. Pour pouvoir recevoir l'information MIDI nécessaire, ces objets, doivent être connectés à un **notein** , qui transmettra toutes les données que vous jouerez, dont certaines seulement seront sélectionnées. Le patch, ci-dessous doit vous imprimer deux **bangs** chaque fois que vous jouerez une des notes en question : le premier correspondra au *Note On*, le second au *Note Off*.



Il faudra donc insérer un **stripnote** entre le **notein** et les sélecteurs pour ignorer les *Notes Off.*

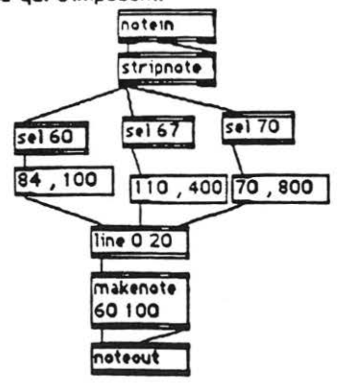



Reste, ensuite à coder les accords. Pour cela il faut créer trois **messages** correspondant aux trois accords. Successivement, 49 et 54 pour le premier, 51 et 57 pour le second, et, 52, 58 et 63 pour le dernier. Un objet **makenote 60 500** (successivement ***vélocité*** et ***durée)*** connecté à un **noteout** assureront la production sonore de ces accords. N'oubliez pas de séparer les *pitches* de chaque accords par une virgule, sans quoi le programme prendra le deuxième argument pour une valeur de *vélocité.* Le patch suivant vous montre la réunification de toutes ces fonctions, réception des données MIDI, sélection, codage des accords et envoi vers le synthétiseur.



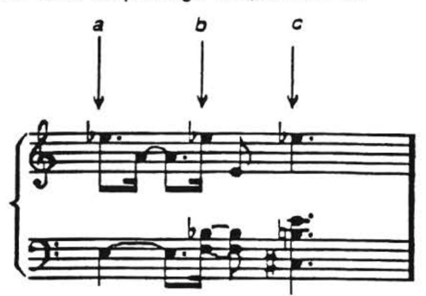

## EXERCICE

Le patch ci-dessous se propose de faire un glissando aboutissant aux valeurs 84, 110 et 70 dans les durées respectives de 100 , 400 et 800 ms. Rappelez vous que l'objet **line** permet de passer, de manière continue, d'une valeur à l'autre. Deux types d'erreurs sont commises qui empêchent la réalisation de cette partition. Cherchez les structures des objets en question et trouvez les solutions qui s'imposent.



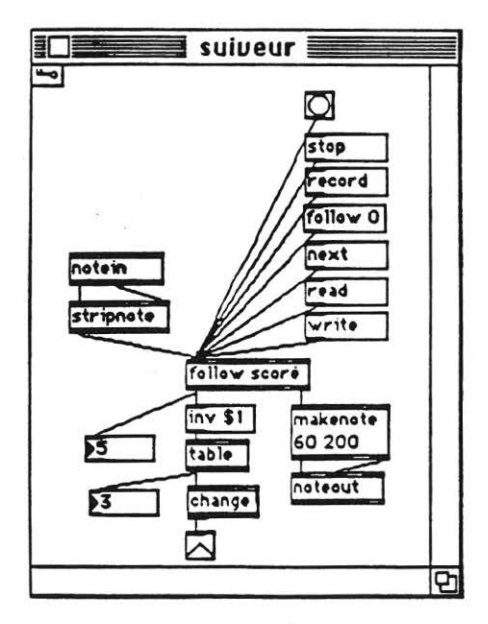
## 4.3. UNE SOLUTION PLUS GENERALE.

L'exemple de détection cité précédemment est, évidemment, très pratique. Mais il est aussi très limité dans le cadre d'une partition plus longue pour la simple raison qu'à chaque fois que vous jouerez une des notes sélectionnée, la même action se produira dans votre synthétiseur.Il n'y a donc pas plusieurs actions possibles à partir d'un même événement, ce qui serait impossible dans un passage tel que celui-ci :




Les trois événements différents *a*, *b* et *c* doivent être déclenchés par la même hauteur *mi bémol*, et le système ne pourra pas reconnaître, avec la procédé que nous avons vu, si le *mi bémol* corresponds à l'événement *a*, *b* ou *c*. C'est pour cela que MAX possède un suiveur de partitions, plus général et plus complet dans son utilisation.

Chargez l'application **suiveur** et apparaîtra une fenêtre identique à celle-ci :




Les objets **notein** et **stripnote** recoivent le informations **MIDI**, que vous fournirez au clavier. Plus exactement, **notein** reçoit des paires *pitch-vélocité* et l'action de **stripnote**

est de les séparer en deux messages différents. Vous constatez que seules les *pitches* sont dirigés vers un autre module (la sortie droite, celle des *velocités* n'est pas connectée). Sur la droite de votre fenêtre vous avez une série de **messages**.

Avant de commencer, il vous faut intégrer ce suiveur de partition à votre propre partition. Pour cela créez un objet **patcher** (<command p>) dans lequel vous créerez un autre **patcher** ou vous copierez l'ensemble de la fenêtre **suiveur**. Rangez **suiveur** et travaillez dans votre patcher qui en est la copie.

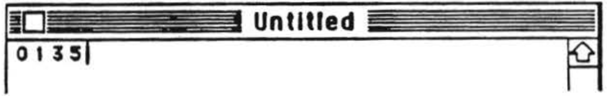
### 4.3.1. L'ENREGISTREMENT DE LA PARTITION

Faites l'expérience suivante. *Cliquez* sur **record** dans la fenêtre que vous venez de créer, qui est la copie de **suiveur**, nommée **untitled**. Cela signifie que MAX est prêt à enregistrer ce que vous allez jouer sur le clavier. Jouez, par exemple, la partie supérieure du petit fragment déjà rencontré:



*Cliquez* sur **follow 0** pour mettre la partition enregistrée au début. *Cliquez* ensuite sur **bang** et vous entendez ce que vous venez de jouer. Le dispositif **makenote** et **noteout** sont, évidemment les agents de cette possibilité de faire rejouer la partition. Les données MIDI que vous avez entrées sont reproduites, mais uniquement dans les durées et les hauteurs. **Suiveur** ne prends pas en compte les *vélocités*. Vous pouvez aussi vous apercevoir que le **number** connecté à la sortie gauche de l'objet **follow score** vous affiche des valeurs : si vous y prêtez attention, vous comprendrez qu'il s'agit du comptage des notes que vous jouez.

Maintenant, que la partition est enregistrée, il vous faut faire savoir au programme sur quelles notes vous voulez placer vos actions. La procédure est la suivante. Ces notes rentrent dans une table qui aura, sur l'axe des *x*, le numéro des notes devant déclencher les actions, et sur celui des *y*, le numéro des actions comptées à partir du début. Cela revient à placer des points précis dans une table. Créez donc un **new** dans lequel vous inscrirez le numéro des notes (reportez vous aux explications concernant les tables pour plus de détails à ce sujet). Attention : *commencez toujours par un 0.* Notre exemple est très simple, les *mi bémol* sont les notes 1, 3 et 5 de la partition, donc :

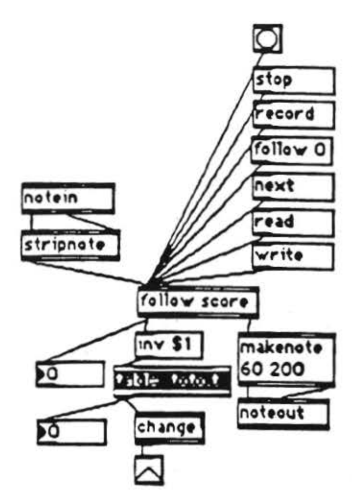



faites ensuite <command c> et, dans la **table**, <command v>, et vos éléments sont entrés. Dans ce cas simple, cela est suffisant, mais dans le cas d'une partition plus complexe, la meilleure solution consiste à jouer, sur votre clavier, la mélodie en question (après avoir *cliqué* sur **follow 0**) en vous arrêtant sur toutes les notes servant de déclencheur, et de lire dans le **number** sélectionné à l'objet **follow score** (qui compte les notes que vous jouez) le numéro auquel elle correspond. Vous inscrirez, ainsi, pas à pas les numéro dans la fenêtre prévue à cet effet (sans oublier le 0 du début).

Une fois cette opération achevée, rejouez la mélodie (toujours après avoir *cliqué* sur **follow 0**), remarquez que le deuxième **number** (celui qui est connecté à l'objet **table**) vous compte les actions. Il doit donc, dans cet exemple vous donner 1, 2 et 3, quand l'autre comptera jusqu'à 5. Tout est prêt, désormais, au niveau du suiveur de partition.

Si vous sortez de MAX, et que vous y entrez de nouveau, il vous faudra réenregistrer toute la partition car la **table** n'a pas été mémorisée. Faites un **save** après avoir sélectionné votre

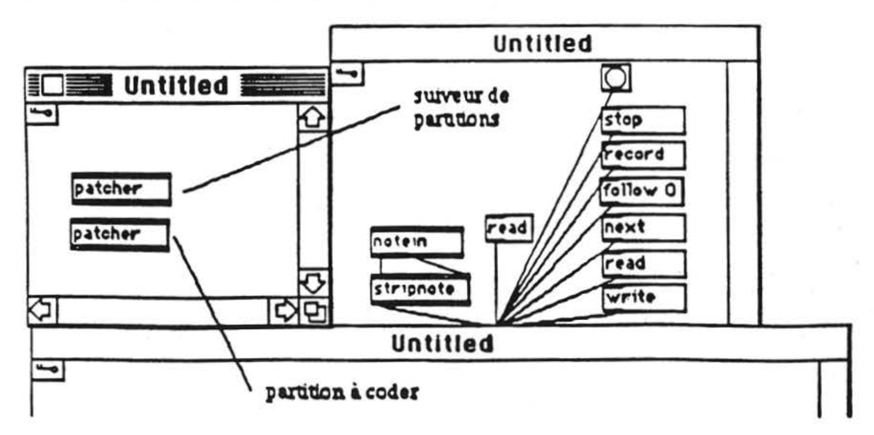
**table**. Le programme vous demande un nom pour la **table**. Nommez la, par exemple " toto.t ". Il faut ensuite ajouter à l'objet **table** le nom " toto.t ". Votre suiveur de partition se présentera comme ceci :




Il faut maintenant faire comprendre au programme que c'est bien cette partition que vous voulez appliquer à ces actions sans avoir à la re-spécifier à chaque fois. Toujours dans votre suiveur de partitions *cliquez* sur **write**. Donnez, comme on vous le demande un nom à cette partition, par exemple " toto.sc " ("sc" pour score). Dès lors, chaque fois que vous *cliquerez* sur **read**, il vous suffira de *charger* " toto.sc " pour installer la partition mémorisée. Un autre moyen, plus rapide, consiste à remplacer **follow score** par **follow toto.sc** ce qui provoquera une lecture automatique de votre partition lors du chargement du patch.

## 4.3.2. LA SYNCHRONISATION DE LA PARTITION AVEC LES ACTIONS.

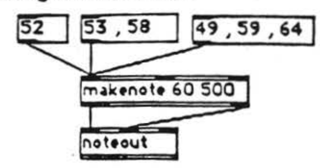
Il faut maintenant coder le reste. Revenez dans votre **patcher** principal, et créez en un autre. La situation est donc la suivante:



Il faut, maintenant donc, lui faire superposer les éléments suivants:



ce qui codé dans des boîtes **message** donnera ceci :

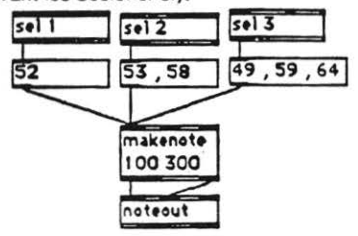



**Un petit conseil pratique:** pour vous éviter de compter les notes sur votre clavier afin de trouver le bon code pour les *pitches*, servez vous de ce petit patch qui vous donnera instantanément les valeurs que vous jouerez au clavier :




Voilà les deux éléments de votre partition réalisés, le suiveur et la partition d'accompagnement. Reste maintenant à faire communiquer ces deux entités entre elles. Elles existent mais ne le savent pas l'une, l'autre. Il faut que ces deux procédures se parlent.

Vous avez remarqué, dans le suiveur de partition, que le **number** issu de l'objet **table** affiche le numéro de l'événement à produire de manière synchrone avec la note. Il faut donc récupérez ce nombre dans votre partition. Pour cela nous utiliserons les ***sélecteurs*** qui sélectionneront les nombres 1, 2 et 3 (c'est à dire les numéros d'apparition des événements et non ceux des notes devant les déclencher).

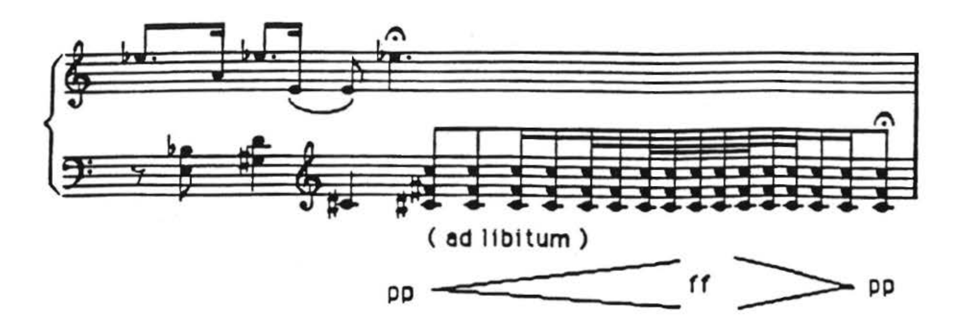



Jetez un **coup d'oeil sur le** patcher "suiveur de partition". Vous voyez que le dernier maillon, en bas, **est un outlet**, qui est d'ailleurs symbolisé par le petit carré noir en bas à gauche de la boîte **patcher :**

Avant d'aller plus loin dans le suiveur de partitions, voici quelques exemples musicaux, tirés du même modèle, qui vous montreront quelques cas de figures à développer.

## 4.4. UN EXEMPLE DE REALISATION.

Soit, l'exemple suivant :




Les problèmes à résoudre sont :

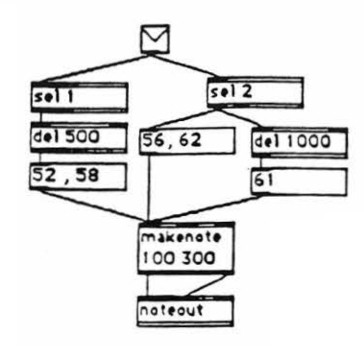
a) les actions retardées
b) l' *accelerrando* et *rallentendo* continus
c) le *crescendo* et *decrescendo*

Le premier cas de figure est très facilement résoluble, il suffit d'intercaler un objet **del** avec la valeur du retard en ms, entre le **sélecteur** et le **message** contenant les valeurs MIDI.
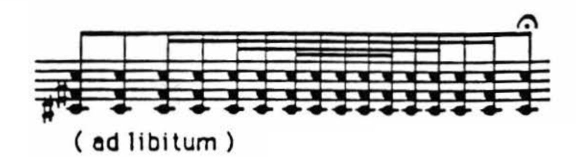
Donc, successivement:




se noteront :

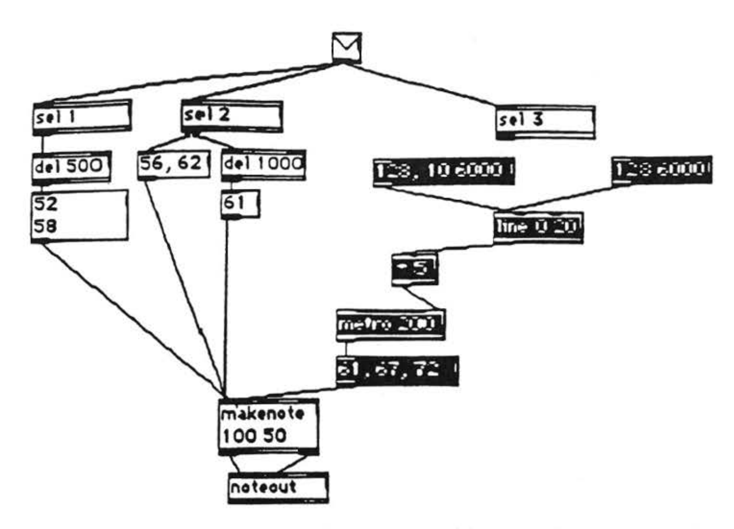



Reste ensuite la dernière formule :

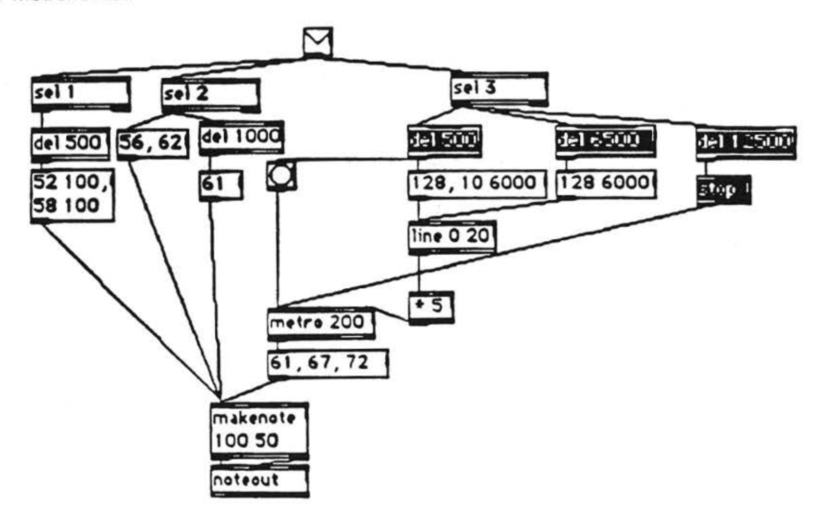



La première précaution à prendre est d'éviter que, dans l'*accelerando*, les durées des notes rapides soient supérieures à celle notée dans le **makenote**, donc changez en **makenote 100 50**. Décidons, ensuite, de donner comme durée à l'*accelerando* et au *rallentendo* la valeur de 6 secondes (6000 millisecondes pour le programme).

Il s'agit finalement de produire un **bang** dans un objet **metro** dont les occurrences seront variables. Pour créer une évolution continue, vous utiliserez un **line 0 20** qui produira une rampe continue entre une valeur minimum et maximum (disons 10 et 128). Le **line** doit être initialisé, c'est à dire qu'il doit comprendre qu'il commencera sa course à 128, puis descendra à 10 en 6000 ms. C'est la raison d'être du message **128, 10 6000**. Le second message remontera à 128 dans la même durée. La valeur qui sort du **line** doit être multipliée par un coefficient qui donnera la valeur en ms des occurrences du métronome. L'objet *** 5** donnera donc 640 ms lorsque la valeur sera au maximum (128 * 5), ce qui équivaut, approximativement à une noire de triolet dans un tempo à 60. Le métronome enverra donc une série de **bangs** au message **61, 62, 72** qui sont les notes de l'accord :

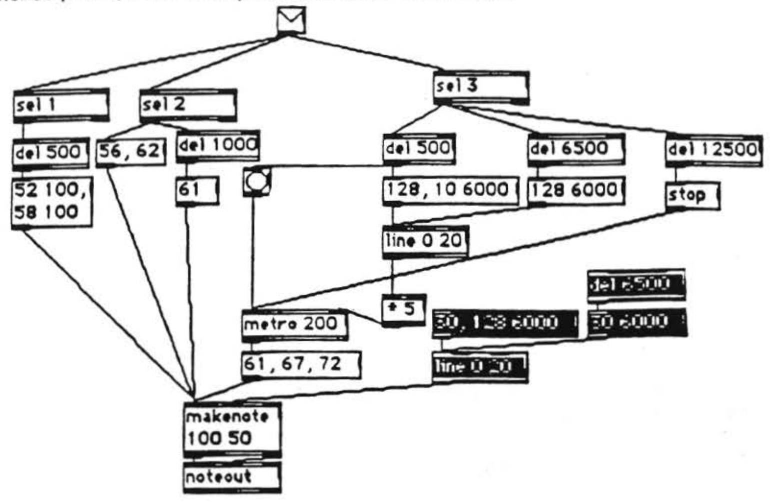



Nous allons installer maintenant notre *rallentendo*. Le processus, comme la montre la partition, commencera avec 500 ms de retard par rapport au dernier *mi bémol*. L'*accelerendo* durant 6000 ms (c'est à dire lorsque la valeur progressera jusqu'au maximum 128), nous commenceront le *rallentendo* avec un retard de 6500 ms. Donc, un **del 500** déclenchera un **bang** de l'objet **metro** ainsi qu'une progression vers 10 en 6000 ms, puis un **del 6500** remontera la valeur jusqu'à 128 toujours en 6000 ms. Enfin, à l'extrême fin du *rallentendo* (c'est à dire avec un retard de 12500 ms) un **stop** arrêtera le métronome:

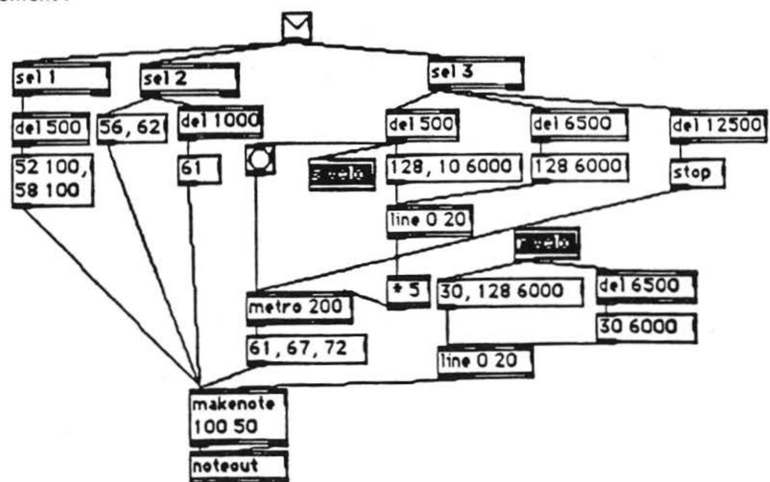



Il ne nous manque que les évolutions d'intensité. Pour ce faire, il faudra agir sur la seconde entrée du **makenote**. Un **line**, là aussi, sera indispensable pour contrôler la continuité de l'effet. On peut facilement imaginer qu'il serait assez mal aisé, pour des raisons de clarté, d'intégrer encore 5 boîtes à la sortie des delais. Il existe un moyen qui nous permettra de

coder ces évolutions dynamiques dans un autre endroit de la fenêtre, puis de les rattacher, aux delais pour le contrôle du temps, et à l'objet **makenote**, pour agir sur les vélocités. Codons d'abord ces évolutions que nous situeront, par exemple, de 30 à 128 pour le *crescendo* puis le contraire pour le *decrescendo* avec un retard identique à celui du *rallentendo* pour que les deux processus soient superposés :

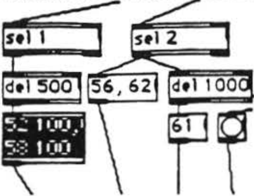



Il faut maintenant rattacher ce processus à l'ensemble. C'est là que vient la grande utilité des **send** et **receive**. Vous créerez un **s velo** qui initiera le *crescendo* à la sortie du **del 500**, ainsi qu'un **r velo** qui continuera le processus jusqu'à la fin. Pour démarrer à la bonne valeur (c'est à dire 30) il faudra envoyer le **message** adéquat dès le début du troisième événement :




Votre partition doit tourner parfaitement si vous n'avez rien oublié. Cependant, si tel est le cas, reprenez la au début. Vous constatez que les niveaux des événements 1 et 2 sont très

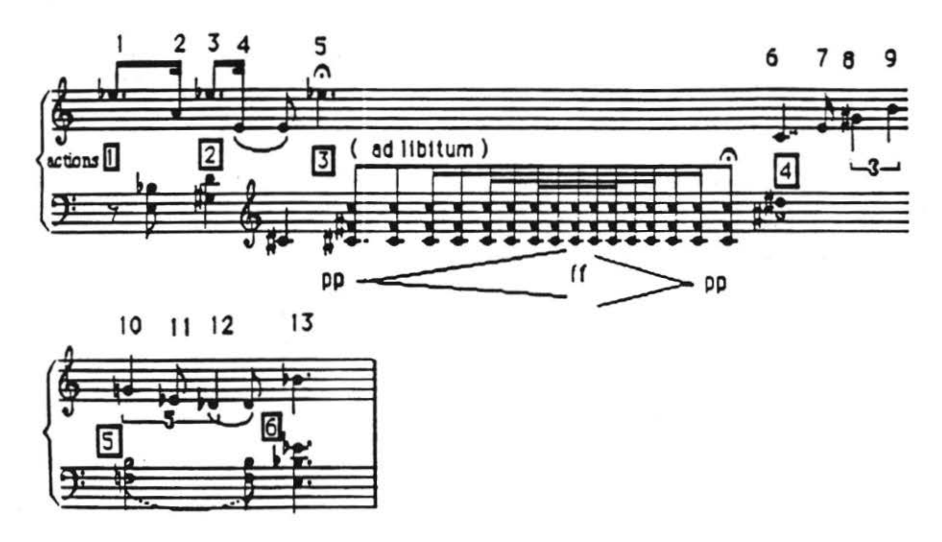
faibles : ils sont restés à la valeur 30. Vous devrez donc leur signifier leur véritable valeur dès le premier événement comme le montre l'extrait ci-dessous :




Cet exemple, vous montrant des styles de programmation, et approfondissant vos connaissance des divers objet que l'on utilise dans le programme, vous révèle surtout une notion importante, celle de *modularité*. On peut, en effet, ajouter des contrôles, comme ici celui des changements dynamiques, de manière indépendante autant que l'on veut sans qu'ils n'affectent les autres processus en place. La partition se monte ainsi, pas à pas, par ajouts de modules successifs.

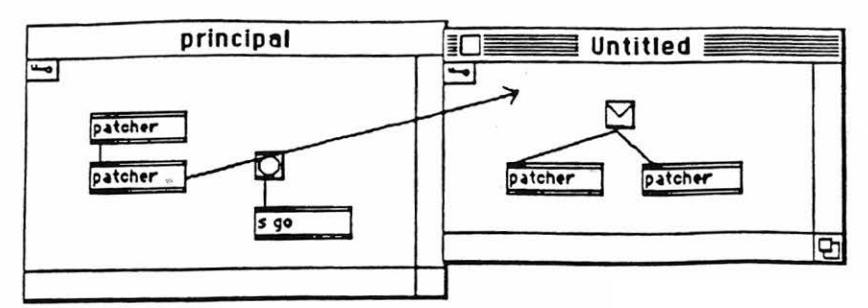
### 4.5. LA SUCCESSION DES PHRASES.

Retournons maintenant au suiveur de partitions. Cette partition est très courte, mais dans la réalité, il vous arrivera de les concevoir beaucoup plus longues. On ne peut pas surcharger les fenêtres indéfiniment sans perdre de la clarté, et nous sommes aussi prisonniers de la taille de l'écran. Imaginons, par exemple, que nous voulons faire succéder les deux partitions que nous avons vu dans ce chapitre. Elle se présentera donc comme ceci :




Il y aura donc 6 événements, les trois du premier exemple devenant les numéros 4, 5 et 6. Les numéros des notes déclenchant les actions seront donc 1, 3, 5, 6, 10 et 13. Enregistrez cette nouvelle partition (cf. plus haut) et chargez les numéros d'événements dans une table. N'oubliez pas de sauvegarder votre nouvelle table.

Comment doit donc se présenter votre nouveau programme ? Tout d'abord, dans votre patch général, vous devrez créer un nouvel objet **patcher** qui sera un objet global et contiendra les deux précédents. Il suffira de faire **cut** et **paste** pour les copier. N'oubliez pas de créer un **inlet** pour pouvoir le raccorder avec le suiveur de partition:
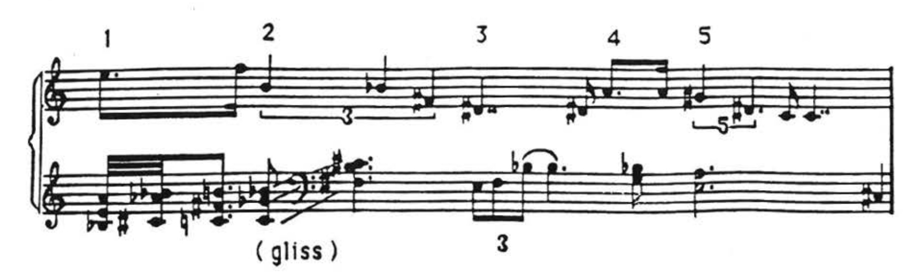



Les deux patchers seront les deux parties de votre partition :

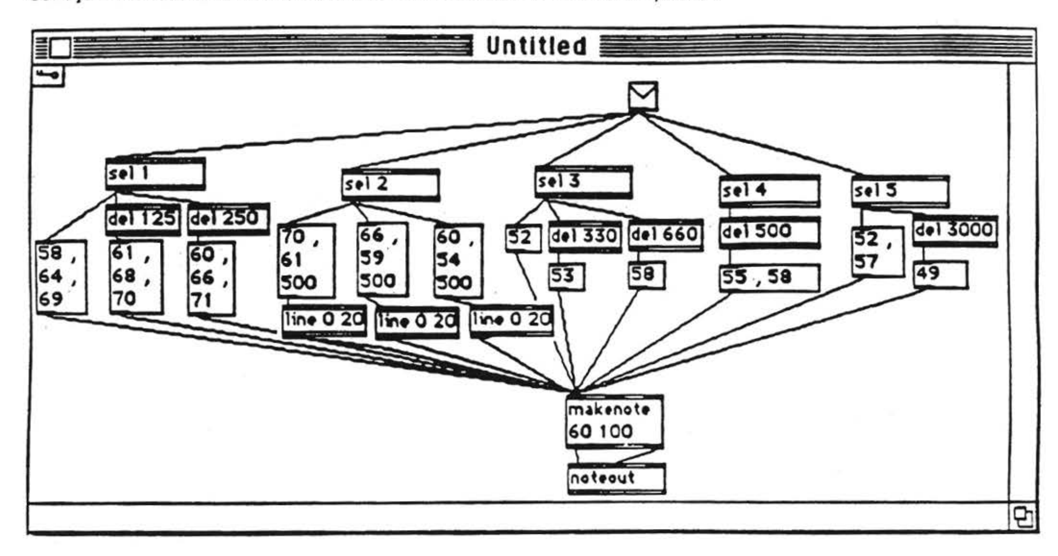
Vous pouvez, ainsi avoir une succession d'objets **patcher** qui contiendront les événements de votre partition et qui seront raccordés à un **patcher** principal.

## 4.6. LE RACCORDEMENT DES SECTIONS.

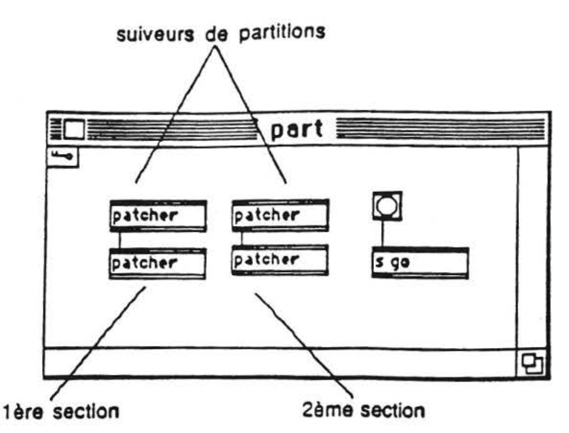
Imaginons maintenant une deuxième section telle que celle-ci :



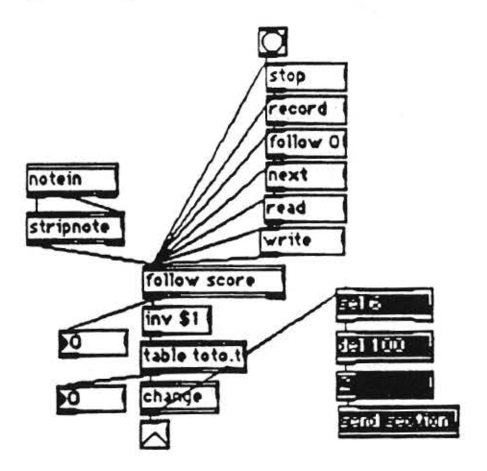

dont je vous donne immédiatement la construction sous forme de patch :




Je pense que vous pouvez maintenant enregistrer la mélodie ainsi que la table sans autres explications. Cependant n'oubliez pas de donner des noms différents à votre partition ainsi qu'à votre table (par exemple : " toto2.sc " et " toto2.t "). Faites ensuite une copie du suiveur de partition qui se raccordera à la deuxième section.Votre programme se présentera désormais comme ceci :

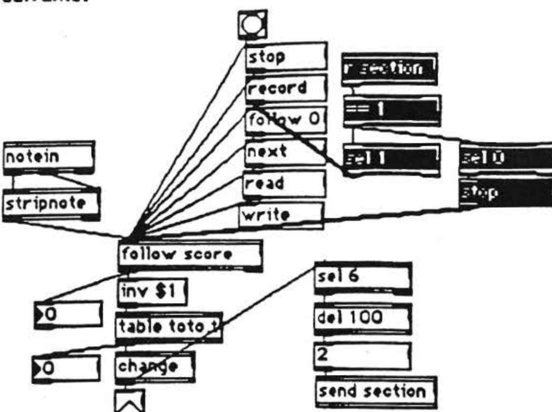



Si les sections communiquent avec les suiveurs de partitions, elles ne communiquent pas encore entre elles. Il est impossible de les enchaîner. C'est ce à quoi nous allons nous occuper maintenant. Il s'agit, en fait, de faire comprendre au programme que lorsque le dernier événement de la première section est lancé, il doit immédiatement être prêt pour la seconde. Pour cela, allez dans votre premier suiveur de partitions et ajoutez la construction suivante à la sortie de l'objet **change** :

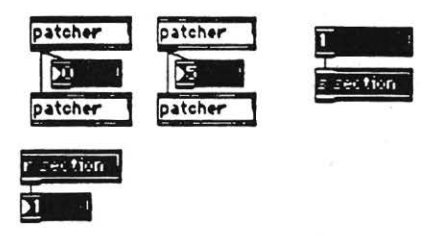



Il y a une nouvelle transmission de messages transitant par la boîte **send section** qui recevra le numéro de la section courante. Dans ce cas le message **2** fera le travail nécessaire. L'objet **sel 6** connecté juste avant la sortie du patch recevra les numéros représentant les actions successives. Celles-ci étant au nombre de 6 dans la première section, le sélecteur déclenchera le numéro de section après un bref retard (**del** 100) pour éviter que la dernière action coïncide avec la première de la section suivante. Jusqu'ici le programme comprend à quel moment il doit changer de section.

Il faut aussi lui faire comprendre quand commence la première section. Ajoutez la construction suivante:



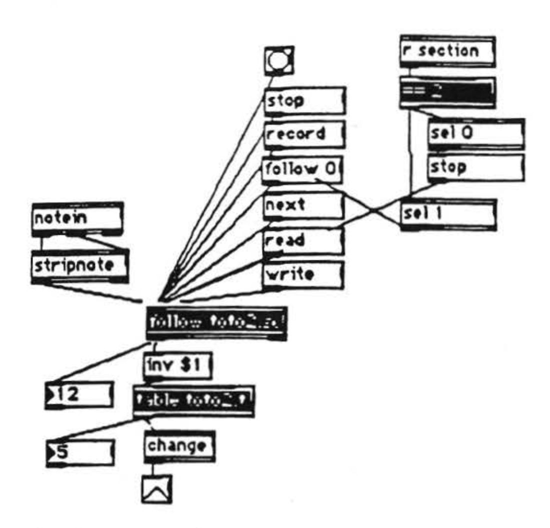

L'objet **== 1** testera une égalité. Si celle-ci s'avère vraie, il retournera **1**, sinon 0. Dans le premier cas (**sel 1**), cela agira directement sur **follow 0**, et à partir de là, vous connaissez la suite. Dans le second cas (**sel 0**) il enverra un **stop** à l'objet **follow score** (ou **follow** <nom de la partition>). Allez dans votre patch principal. C'est à partir de celui-ci que vous enverrez le message **1** à l'objet **section** pour le positionner au début de la partition.




Remarquez plusieurs changements. Un objet **r section** débouchant sur un **number** vous indiquera la section courante. Un **number** connecté à chaque suiveur de partitions, vous indiquera aussi l'événement courant dans chaque section. Donc pour démarrer le programme, il suffira de ***cliquer*** sur le message **1** envoyé à **s section**.

Si la première section sait comment s'enchaîner à la deuxième, celle-ci doit avoir les indications nécessaires pour comprendre les messages qu'elle reçoit. L'exemple suivant vous montre le suiveur de partition de la seconde section et les changements intervenus par rapport à la première. D'abord, le numéro de la section dans le test d'égalité doit correspondre à la section courante (**== 2**). Ensuite, comme je vous l'ai déjà dit, il faut modifier le nom de la table. Allez dans votre second suiveur de partitions et modifiez-le en conséquence. Vous constatez que le nom de la deuxième table ainsi que celui de la deuxième partition ont été inscrits. Profitez en pour faire de même dans la première section, soit :

suiveur de partition 1: **toto.sc** et **toto.t**
suiveur de partition 2: **toto2.sc** et **toto2.t**




**Attention:** lors du chargement de votre patch, les partitions seront lues automatiquement sans avoir besoin de *cliquer* sur **read**. Pour cela, elles doivent être **obligatoirement dans le même dossier** (*folder*) que votre patch principal. En principe tout doit marcher parfaitement si vous n'avez rien oublié. Si vous avez beaucoup de difficultés, regardez l'application **partition** qui réalise ce modèle. Vous connaissez maintenant les principales fonctions du suiveur de partition. Mais il y a quelques commentaires à faire à son sujet.

## 4.7. LE PROBLEME DES ERREURS D'EXECUTION.

Je devine la question que vous vous posez certainement : que se passe-t'il en cas de fausses notes ? Il y a plusieurs degrés dans l'erreur, les erreurs rythmiques, on l'a vu, sont tolérées. Heureusement d'ailleurs car c'est dans ce domaine que l'interprétation offre le plus de variabilité. Mais dans quelle mesure les erreurs de hauteurs sont-elles tolérées ?

Si j'intercale, en reprenant notre exemple, diverses notes entre les notes principales de mon fragment, le programme n'aura aucune gêne à reconnaître celles qui étaient mémorisées, il ignorera simplement les autres :



Par contre, si je remplace ou omet une des notes enregistrées comme ci-dessous :
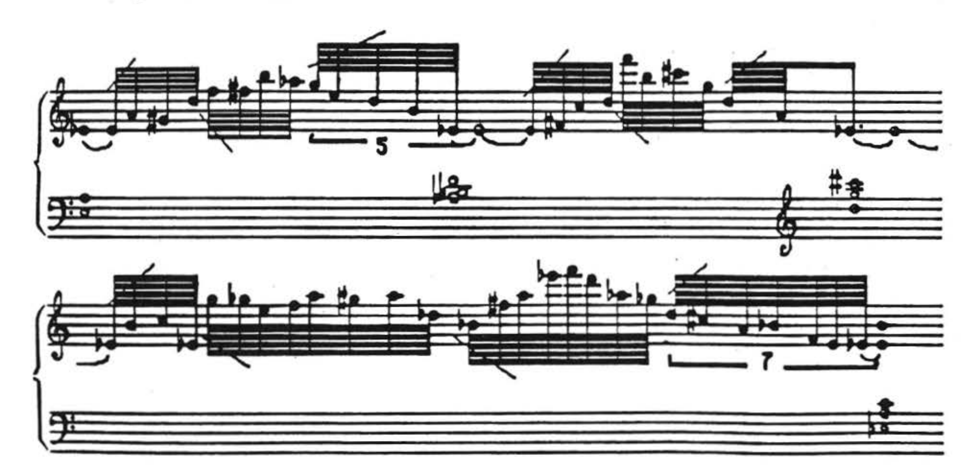


alors, le programme attendra jusqu'à ce que cette note apparaisse. Mais cet exemple était relativement simple. Pour être général, un tel programme se doit de pouvoir suivre n'importe quelle partition pourvue qu'elle lui fournisse des données MIDI. Et dans le cas de partitions plus complexes, ou qui demande une assez grande virtuosité, une erreur n'est jamais exclue.

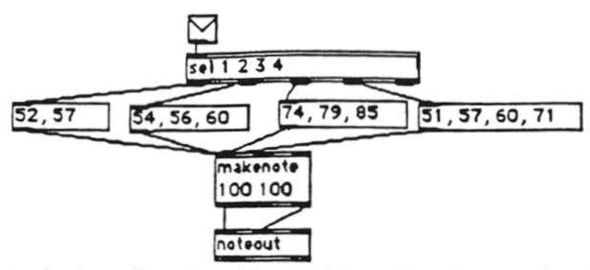
Tel qu'il vous a été présenté ici, le suiveur de partitions exige une parfaite exécution des notes, ce qui, dans le cas précédent reviendrait à un arrêt brutal des actions devant être envoyées au synthétiseur. Dans un concert: une catastrophe.

Il y a donc un moyen de tolérer certaines erreurs et de rendre le programme plus ou moins tolérant. Un clavier, par ses possibilités, est capable de fournir une partition d'une très grande complexité due à la polyphonie et aux diverses superpositions. Examinons le cas.

Soit ce fragment à réaliser :



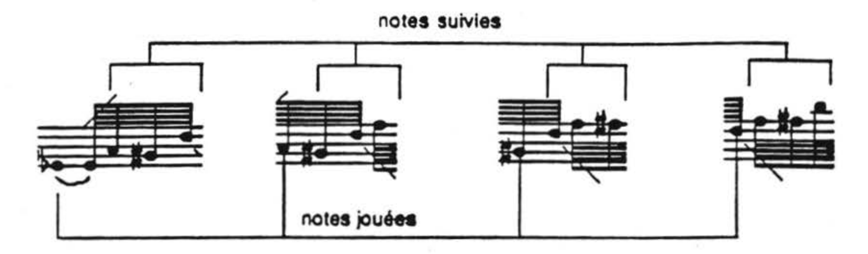
Le codage de ces quatre événements est très simple:
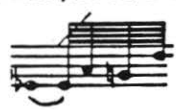



(l'objet **sel 1 2 3 4** équivaut à quatre objets **sel1**, **sel2**, **sel3** et **sel4**, mais comporte une sortie de plus que le nombre d'arguments).

C'est au niveau du suiveur de partition que va se jouer le niveau de tolérance. Mais d'abord, posons nous la question de savoir comment fonctionne celui-ci.

Imaginez-vous une fenêtre qui va sélectionner les notes quatre à quatre. C'est à dire que lorsque vous jouerez la première note, il regardera les trois suivantes et ainsi de suite comme vous le montre l'exemple ci-dessous :




Si, vous ne jouez pas la bonne note, et que le suiveur de partitions soit programmé avec une certaine tolérance, il ignorera cette erreur pourvu que ce que vous jouez ensuite soit correct. C'est à dire que lors de votre erreur, il fera ce que l'on appelle, dans le jargon informatique, un *pattern matching*. Voyons, en quoi cela consiste. Prenons la première figure ci-dessus. Le suiveur de partition a mémorisé (*mi b, la, sol #, ré*, soit en code MIDI : 63 , 69 , 68 et 74). Imaginons que vous jouerez, à la place de cette figure, la suivante:



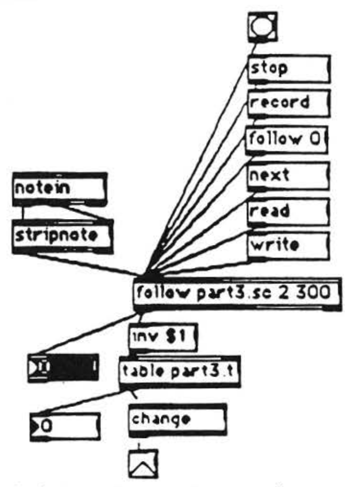
il fera alors une comparaison entre ces deux patterns :




et , si je peux m'exprimer ainsi, tiendra à peu près ce langage.

> Entre ce que j'attendais et ce qui m'est parvenu, s'est produit une erreur sur le "sol #". La dernière note connue au moment de l'erreur est donc le "la". Mais lorsque je reçois qui est la note suivante dans le pattern la dernière note connue est maintenant le "ré". Or, je connais le "ré", donc j'oublie le "sol bécarre", et je continue.

Ce raisonnement "anthropomorphique" est bien sûr une image de la réalité. La suite de décisions que prends le programme n'a pas ce niveau d'intelligence, mais le résultat peut le simuler. Mais, comme il ne s'agit pas ici de faire une analyse sémantique, cet exemple parviendra, j'espère, à vous faire comprendre le fonctionnement du suiveur de partitions. L'expérience a prouvé, dans ma pièce "*Pluton*", qui pourtant est d'une assez grande difficulté d'exécution, que le seuil de tolérance peut être réglé sans trop de problèmes pourvu que l'interprète soit évidemment très près du texte écrit, sur deux erreurs consécutives. Au dessous, ce serait trop risqué, au dessus, cela poserai d'autres problèmes de reconnaissance. Faisons en donc l'expérience. Chargez l'application **partition2**, et *cliquez* sur **follow 0** dans le suiveur de partitions.




Le **number** sélectionné, ci-dessus, je vous le rappelle, compte le nombre de notes. Jouez, par exemple, le début, tel qu'il est écrit :



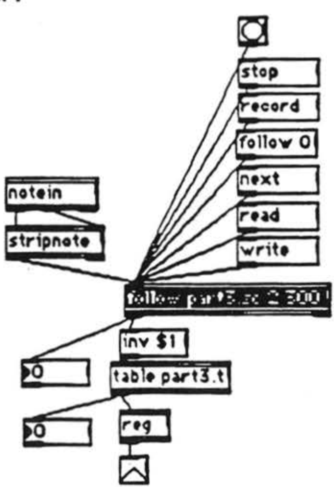
arrivé au *si*, le **number** doit vous donner 7. Jouez maintenant le passage suivant avec ces deux erreurs consécutives:



le **number** , après s'être figé sur le 3 (*sol#* étant le dernier événement reconnu), sautera directement sur 6 lorsque vous jouerez le *fa#* (il ignorera 4 et 5), et vous donnera encore 7 sur le *si*. Maintenant, si à la place du *fa#*, vous jouez un *sol*, par exemple, donc 3 erreurs consécutives, le suiveur sera perdu. C'est, direz vous, encore assez dangereux, mais, en y réfléchissant, la probabilité de faire plus que deux erreurs consécutives doit être à peu près nulle pour un bon exécutant. Ceci dit, il n'y a pas qu'une manière d'entrer les partitions dans le suiveur. Très souvent, c'est sous une forme simplifiée qu'il faut les mémoriser. Mais nous verrons cela un peu plus tard.

## 4.8. LE PROBLEME DES ACCORDS.

Il y a des cas, ou certaines configurations musicales posent des problèmes de suivi. C'est, en particulier le cas des accords. Qu'est-ce qu'un accord sinon, théoriquement dans la partition, que la production de plusieurs notes d'une manière simultanée. En fait, il faut s'interroger, quelques instants sur cette notion de simultanéité. Un tel système est capable de reconnaître les événements à la milliseconde près. Or, ce qui pour l'oreille est simultané, par exemple, lorsque l'on joue un accord sur un clavier, ne le sera pas pour ce système. Il ne doit pas y avoir de confusions entre mélodies et accords. Lorsque vous jouez un accord, il y a un décalage entre les notes, c'est finalement un arpeggio très rapide que vous produisez. Pour éviter cette confusion, il est nécessaire, là aussi, de fixer un seuil en dessous duquel, tout ce qui est *successif*, sera reconnu comme *simultané*. On peut le fixer, raisonnablement à 300 ms. C'est la raison pour laquelle dans le suiveur de partitions de cette application, les trois arguments de **follow** sont *<nom de la partition>* 2 300. Le premier argument faisant référence au nombre d'erreurs tolérées, le second au seuil de simultanéité. Ils figurent ici pour information, mais sachez que ces valeurs sont prises par défaut, donc il n'est pas nécessaire de les spécifier :
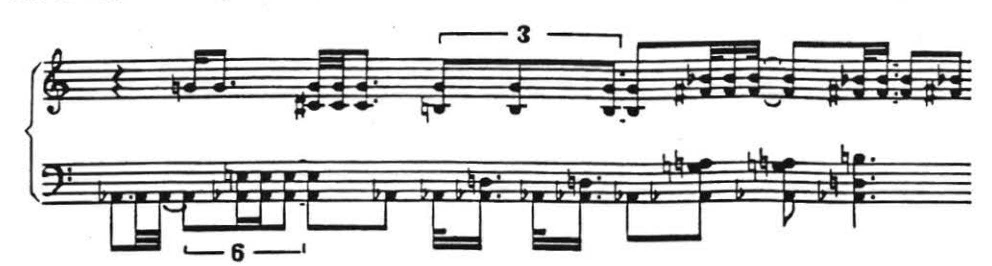


## 4.9. DES CONFIGURATIONS PROBLEMATIQUES.
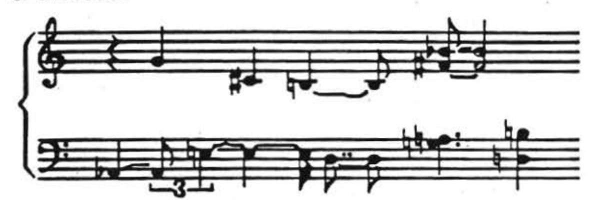
Il y a des cas, comme je l'ai écrit plus haut, ou il est nécessaire de mémoriser les partitions sous une forme simplifiée. Nous allons en examiner quelques uns.

### 4.9.1. Les notes répétées.

Soit un fragment tel que celui-ci :
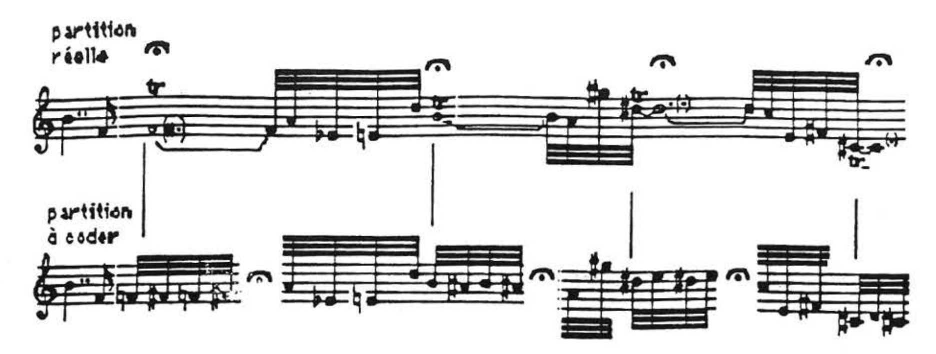



Les risques d'erreurs sont grands, car la répétition à très grande vitesse n'est pas assurée. Il faut, en tout cas, que le système obtienne un *note off* avant de reprendre à nouveau la note et un accident serait vite arrivé dans ces conditions si la partition était mémorisée telle quelle. La solution la plus économique, et aussi la plus sûre, est d'entrer simplement les changements de notes, en ignorant tout simplement les répétitions comme le montre l'exemple ci-dessous :



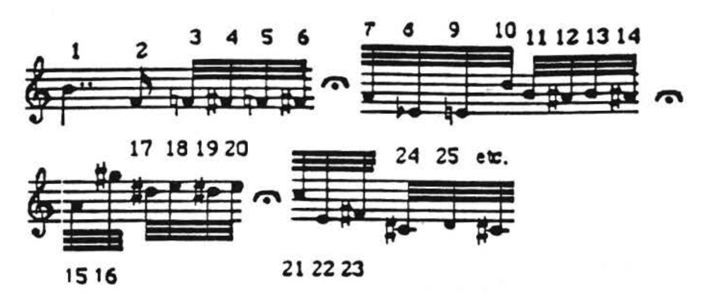

J'ai, ici, respecté les durées séparant chaque nouvelle entrée, mais il est suffisant d'entrer les notes de manière plus approximative, pourvu que soit respecté la bonne succession des nouvelles hauteurs. Codez cette partition et testez là pour vous convaincre de son bon fonctionnement.

### 4.9.2. Les événements en nombre indéterminé.

a) Je fais entrer dans cette classe d'événements, les trilles, battements, trémolos ou tout autres figures dont le nombre des notes est fonction de l'interprétation du moment, et ne peut être prédéterminé. Le fragment suivant vous montre la partition réelle, avec les trilles et les points d'orgue, puis la version qu'il vous faudrait coder pour la mémoriser :

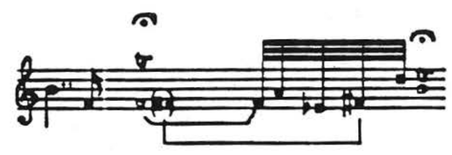



Que se passe-t'il en fait, au niveau du suiveur de partitions? Il comptera les événements comme je l'indique ici :
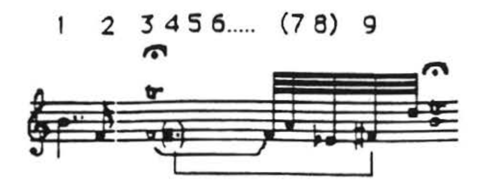



Ainsi, lorsque vous jouerez, par exemple le premier trille, il attendra le "*la*" qui est le septième événement et ne bougera pas pendant que vous continuerez à jouer " *fa, fa#, fa, fa#* etc .. " . Vous pouvez, effectivement, mémoriser plus de notes du trille, le risque consistant, lors de l'exécution, à jouer moins de notes que ce qui à été mémorisé. Donc, en règle générale, lorsque vous aurez à coder des événements stables et en nombres indéterminés, le codage des premiers événements sera suffisant.

b) Cependant il nous faut examiner un cas impossible à traiter pour des raisons aisément compréhensibles compte tenu du seuil de tolérance de deux erreurs consécutives que l'on s'est fixé. Si une des notes du trille apparait dans l'une des trois notes suivantes comme dans l'exemple ci-dessous :




le suiveur interprétera les notes "*la mib* " comme deux erreurs consécutives par rapport à une note du trille qui serait un *fa bécarre* normalement attendu, et il comptera les événements comme suit :




c'est à dire que les événements 7 et 8 seront ignorés (car considérés comme faux) et le "*fa#*" qui suit sera pris comme neuvième élément.

c) Voici, pour terminer sur ce chapitre, comment il faudrait coder un passage tel que celui-ci :

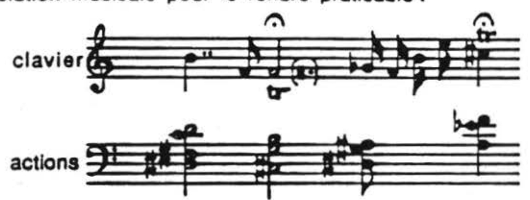


pour éviter les types de problèmes évoqués précédemment, a savoir le retour des éléments du trille (ici dans la reprise du trille) il sera suffisant de coder cette version simplifiée :



**EXERCICE :**

Le petit fragment suivant comporte une impossibilité. Comprenez-en la raison, et tâchez de trouver une solution musicale pour le rendre praticable :



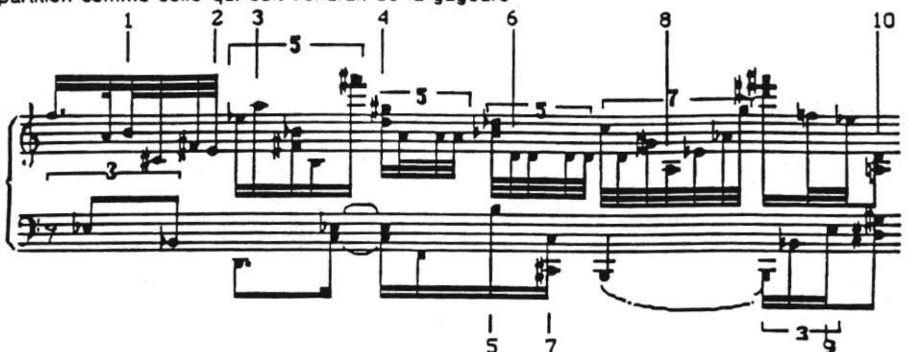

**4.9.3. les passages de grande virtuosité (ou les risques d'erreurs sont assez probables).**

Il serait vain de prétendre que, dans notre musique, jouant parfois sur une virtuosité extrême, les meilleurs interprètes ne produisent pas d'erreurs. Le tout est de savoir si l'erreur est de nature à remettre en question le style même de la musique qui se joue. C'est là une position qui a ses adeptes comme ses détracteurs. Ce n'est pas le lieu, ici, de débattre de telles questions, bien que, personnellement, je sois de plus en plus favorable à une adéquation entre ce qui est écrit et ce qui doit être joué. Mais, il faut bien reconnaître qu'une erreur de note dans "*Herma*" de Xénakis, ou même dans certains passages du "*klavierstuck X*" de Stockhausen ou encore de la "*seconde sonate*" pour piano de Boulez peut passer tout à fait inaperçue même pour les oreilles les plus exercées.

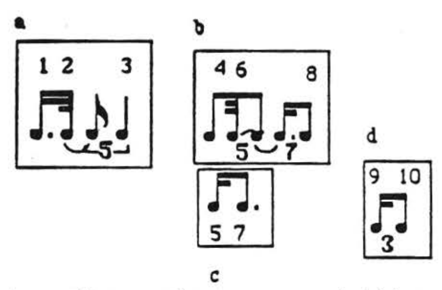
Un système tel que celui dont nous nous occupons en ce moment n'a pas le pouvoir, lui, de juger une qualité esthétique par rapport à un style donné, ou bien ce prétendu style serait d'une pauvreté désarmante. Cependant, il est sans rival pour détecter instantanément les erreurs. Et, nous ne nous le répèteront jamais assez, une fausse note, même jugée acceptable par l'oreille, peut provoquer une catastrophe si elle est responsable du blocage du suivi de partitions. Il est évident que, dans ma pièce "*Pluton*", les passages virtuoses ne manquent pas. Il est aussi évident que j'ai gardé la très haute virtuosité pour des sections ou le

synthétiseur n'avait pas à entrer en jeu. Tout le travail de composition avait entre autre pour but, de trouver un juste équilibre entre des partitions *suivables* et d'autres qui ne l'étaient pas sans grands risques, sinon l'altemance entre des passages *complexes* "solo", et d'autres, *simples* "avec le synthétiseur", serait vite devenue très artificielle. Il n'en demeure pas moins que, voulant éviter au maximum toute intervention extérieure qui se chargerait de "caler" le suiveur de partition à un moment donné du jeu du pianiste, il fallait bien que ce système puisse reconnaître des événements virtuoses, ne serait-ce que pour pouvoir rentrer au bon moment, même s'il n'avait pas d'interventions prévues à cet instant.

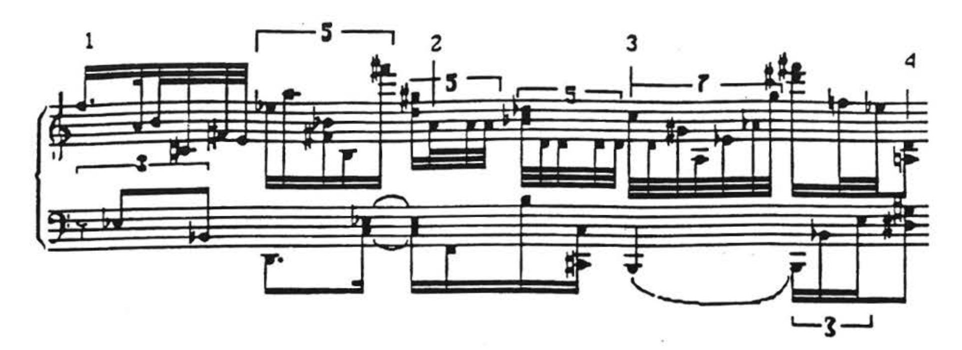
On le voit, et ceci est un facteur très important, les méthodes appliquées aux problèmes de suivi de partitions demandent avant tout une bonne intelligence du fonctionnement de celui-ci par rapport à une situation donnée. Il n'y a pas de recettes miracles là où la performance humaine peut faire défaut. Il est évident que distribuer dix actions successives dans une partition comme celle qui suit tiendrait de la gageure



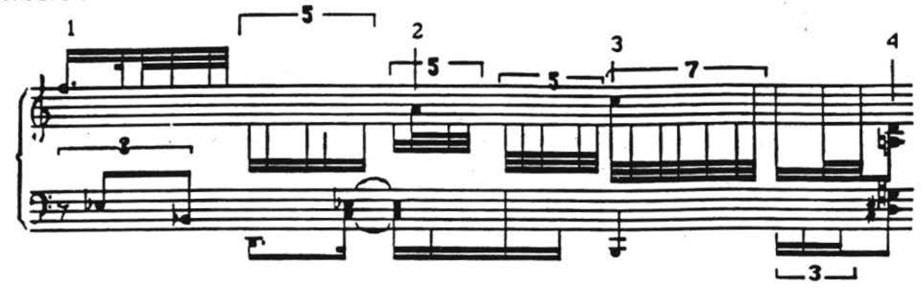

et que la solution suivante, consistant à regrouper le tout en quatre actions a, b, c et d qui seraient programmées avec des retards limiterait considérablement les risques d'erreurs même si certaines actions ne tombent pas exactement sous certaines notes :




Mais, en admettant que votre partition se présente avec ces 4 objets :
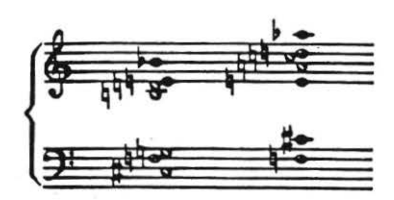



alors la mémorisation de la partition filtrée comme suit sera suffisante et évitera les erreurs :



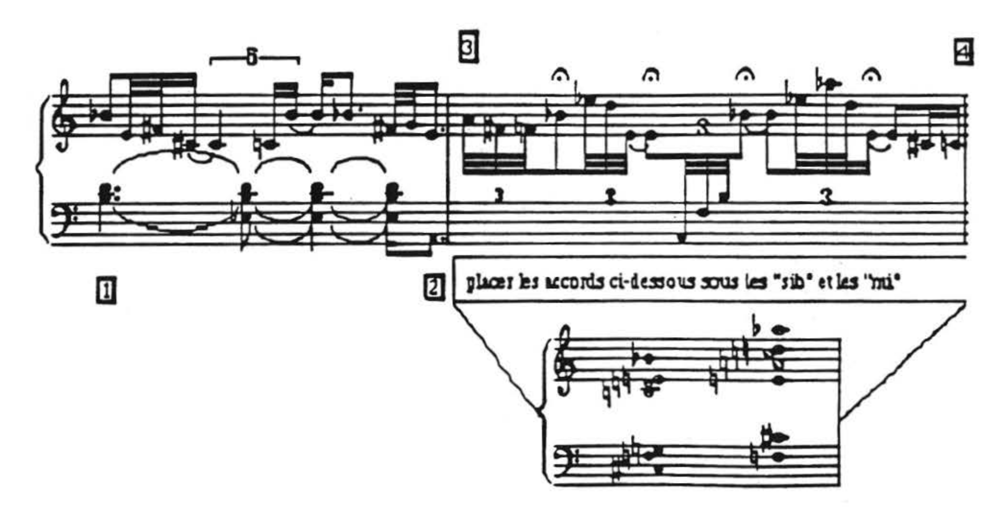

Il faut finalement s'assurer qu'il ne peut y avoir de confusions de notes (par exemple le deuxième élément est déclenché par un "*la*" qui apparait déjà une première fois au début donc il doit y avoir au moins trois notes entre ces deux "*la*" ), et tâcher de coder les événements que l'interprète ***est sûr de ne pas rater***, d'une part, et, d'autre part, s'assurer que la note attendue ne figure pas plus tôt dans la partition.
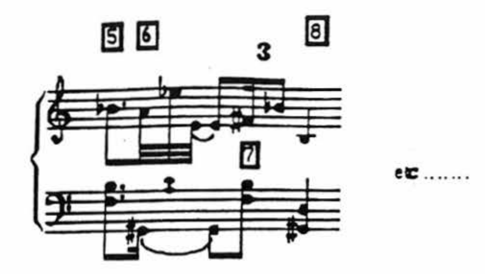
## 5. LES ACTIONS LOCALES

J'entends par " action locale " toute action ne devant être exécutée que si certaines conditions précises sont remplies. Il peut arriver souvent que l'on désire obtenir un contrôle quelconque sur le synthétiseur pendant le temps d'une section, et que ce contrôle disparaisse ensuite. Je donne un exemple concret. Vous désirez, que soit exécuté un accord type, chaque fois que vous jouerez un *sib* et un *mi*, par exemple :



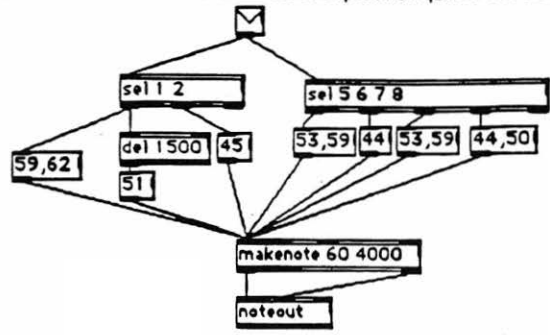
mais vous voulez aussi pouvoir utiliser ces deux notes dans un contexte différent sans que ces deux accords n'apparaissent. Il faudra donc créer les conditions nécessaires qui feront comprendre au programme qu'il doit entrer dans ce cas de figure, puis en sortir. Examinons le cas en donnant une partition qui repose justement sur ces deux notes, en tant que "notes-pivots", mais dont les fonctions soient changées en cours de route. Voici la partition :



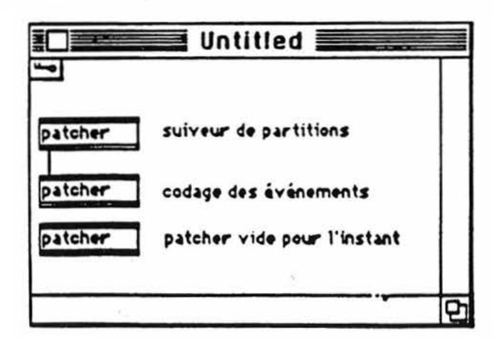



etc.......

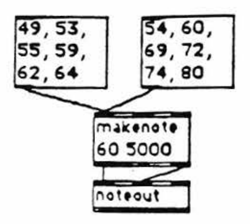
Entrez la mélodie dans un suiveur de partitions, chargez la table des événements, puis connectez votre patcher à celui du codage des événements comme nous l'avons fait jusqu'à présent. Analysons très rapidement cette mélodie. La première et la troisième mesure proposent un schéma classique que nous avons vu souvent, c'est à dire, des accords qui se synchronisent avec la mélodie. La seconde, par contre, doit ignorer toutes les notes, sauf les *sib* et les *mi* qui déclencheront les accords. Ces accords, d'autre part ne devront pas apparaître dans les mesures 1 et 3. Le codage de ces deux mesures (c'est à dire des événements 1, 2, 5, 6, 7 et 8)est indiqué ci-dessous et se trouvera donc dans le patcher qui lui est consacré :



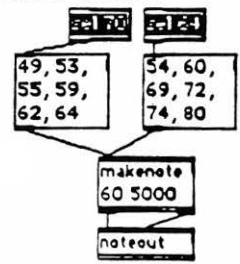

Reste, maintenant, celui des événements 3 et 4 qui demande des précautions supplémentaires. Il s'agira de définir la condition nécessaire à ce processus d'accords. Cette condition est finalement donné directement dans la partition lors du déclenchement de l'événement 3. Comment procéder alors. Créez un nouveau patcher dans votre patcher principal qui se présentera sous cette forme -ci :



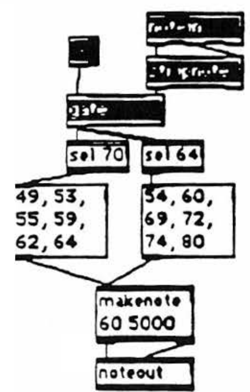
Le nouveau **patcher** contiendra les informations nécessaires à la réalisation de ce processus. Commençons par coder les événements eux mêmes, c'est à dire les deux accords:



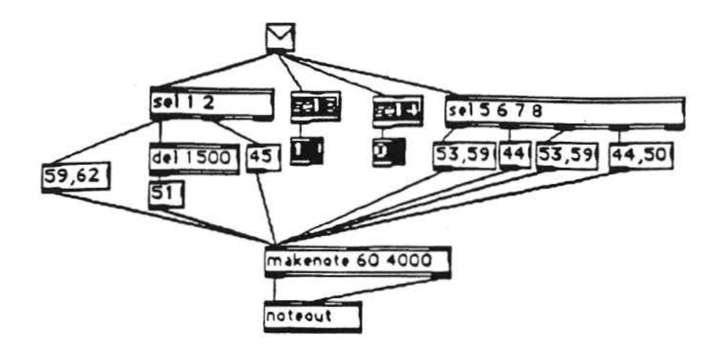

La sélection de ces accords se faisant sur reconnaissance des *sib* et *mi* (soit en notes MIDI 64 et 70), les **sélecteurs** feront l'affaire :



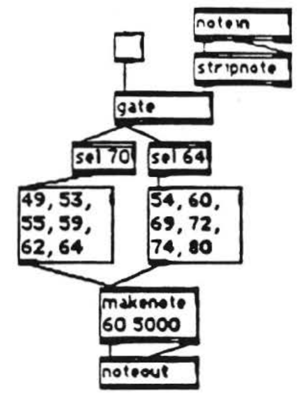

C'est maintenant qu'il va falloir indiquer au programme les conditions d'ouverture de ce processus. Pour cela il va falloir faire passer l'information par une "porte" qui sera ouverte, ou fermée, suivant que la condition sera remplie ou non. L'objet nécessaire à cette action est un **gate** qui, connecté à la sortie d'un **toggle** donnera 0 (dans la position fermée) ou 1 (dans la position ouverte). L'entrée droite du **gate** recevra, quant à elle les informations du clavier au moyen d'un **notein** et d'un **stripnote** pour ne pas produire d'actions au moment d'un *Note Off*, sinon l'accord se répètera aussi lors du relâchement de la touche lorsque vous jouerez les notes en question :



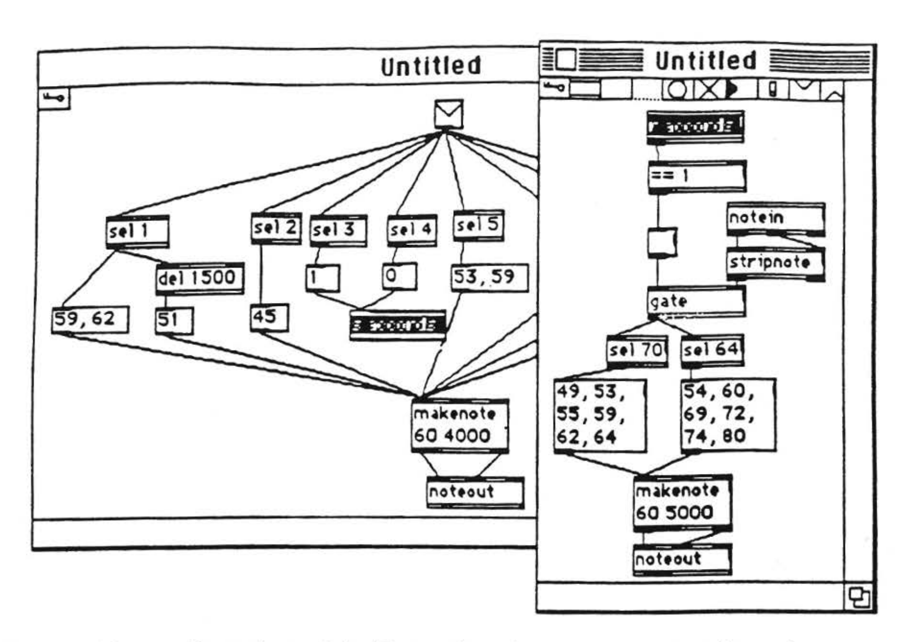

L'entrée du **gate** recevra donc une information de la partition, si cette valeur est égale à 1, alors on ouvrira la porte, si elle est égale à 0, on la fermera. Ces deux valeurs se trouveront envoyée lors des événements 3 et 4 dans des **objets** :




Le test qui vérifiera si la valeur est égale à 1 ou 0 ira ensuite à l'entrée du **gate** et ouvrira ou fermera le processus suivant le cas. A savoir, lors du troisième événement, la valeur 1 sera envoyée au **toggle** situé à l'entrée du **gate** (nous verrons comment dans un instant) et ouvrira la condition de sélection des *pitches* 70 et 64 (à savoir *si bémol* et *mi*). Lors de la sélection du quatrième événement, le **toggle** recevra la valeur 0 et fermera cette condition. Il faut évidemment veiller à ce que la note qui déclenche le quatrième événement ne soit pas une de celles qui figurent dans ces conditions locales, sinon il sera impossible de sortir de ce cas de figures.
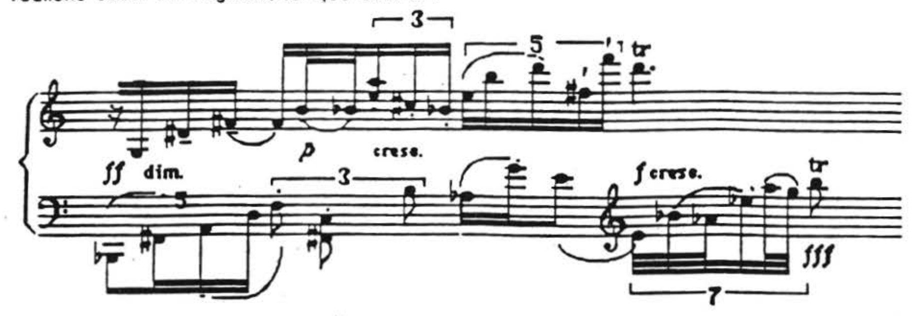



Enfin, il ne restera plus qu'à connecter les deux actions, celles envoyées par la partition, et celles reçues par le patcher comprenant ce processus. Pour cela nous utiliseront les messages, par un **send accords** dans la partition qui passera les messages 1 et 0, et un **receive accords** qui les transmettra à l'entrée du **gate** :




Votre partition est désormais terminée. En cas de panique, vous pouvez toujours charger l'application **local** qui réalise exactement ce cas de figures, et tâcher de comprendre quels sont les erreurs que vous avez commises dans l'élaboration de votre patch.

## 6. LE CODAGE DES SEQUENCES

Nous avons vu, jusqu'ici, le codage d'événements assez simples, comme des accords, des notes isolées, des figures au rythme relativement simple. Mais imaginons que nous voulions coder un fragment tel que celui-ci :

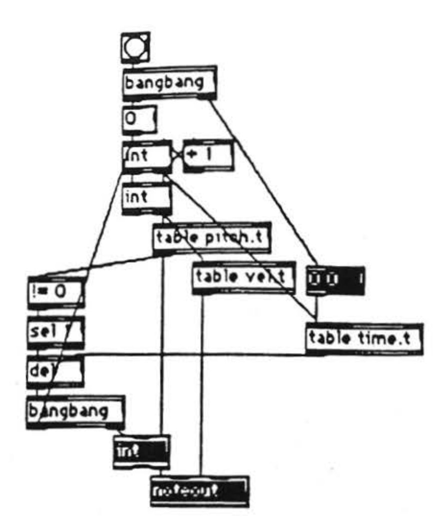



il serait extrêmement malaisé de le coder, en valeurs de retards (avec des **del**), et le calcul des différentes valeurs rythmiques deviendrait vite fastidieux. Il existe un moyen bien plus sûr d'y parvenir, c'est d'utiliser un **sequencer**, qui vous permettra de mémoriser ce que vous voulez obtenir. Et la meilleure possibilité de rentrer ces données est évidemment, pour peu que vous soyez assez bon pianiste pour jouer un fragment tel que celui-ci (sinon adressez vous à quelqu'un qui le fera pour vous), de les jouer directement sur votre clavier. Seront pris en compte, les hauteurs, rythmes, durées et vélocités. Bref, le sequencer vous reproduira le fragment exactement tel que vous l'avez enregistré. Le maniement de ce module est extrêmement simple et il n'est pas besoin d'y consacrer tout un chapitre. Seulement, le patch réalisant ce sequencer est assez difficile, et je pense que c'est une bonne occasion pour vous montrer une réalisation plus complexe.

### 6.1. UN EXEMPLE DE SEQUENCER

Chargez l'application **sequencer**, et apparaîtra le patch en question. Il y a deux **bangs**, l'un pour enregistrer, l'autre pour reproduire une séquence donnée. Actionnez, donc le premier (celui de droite), jouez une séquence sur votre clavier, puis actionnez le second (celui de gauche), et votre séquence sera exécutée exactement avec les mêmes valeurs de durées, rythmes et vélocités.

Essayons de comprendre comment cela fonctionne en fait. Pour commencer, vous allez étudier la partie droite de ce module, c'est à dire celle qui se charge de la mémorisation de votre partition. Le procédé consiste finalement à inscrire des valeurs dans trois tables différentes, **pitch.t**, **time.t** et **vel.t** correspondantes aux trois composants que vous voulez mémoriser. Une **table**, je le rappelle, est un espace à deux dimensions *x* et *y* dans lequel seront stockés les valeurs (sur l'axe des *y*) et leur ordre d'apparition (sur celui des *x*). Les deux **inlets** d'une table correspondent donc à ces deux axes (*x* à gauche et *y* à droite). Voici comment s'inscrivent les valeurs dans les trois tables:

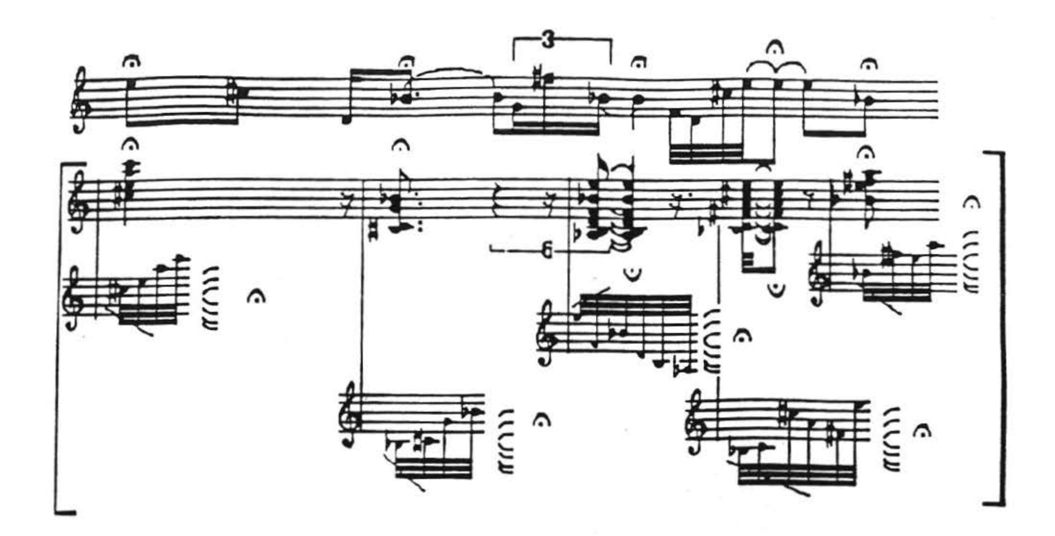



les valeurs lues dans les tables transitent par **int**, qui reçoit les valeurs de *pitch* de la table et un **bang** produit par le **del**, puis entrent dans le **noteout** pour la synthèse. D'autre part, le message " 0 0 " permet de mettre une valeur nulle à l'entrée de la table des durées. En effet, la valeur initiale, était celle qui séparait le **bang** d'enregistrement de la première note que vous avez joué. Sans cette précaution, lorsque vous auriez actionné le processus de reproduction de la séquence, vous auriez obtenu, du même coup, une première valeur égale à ce temps. Ainsi, vous pourrez synchroniser votre séquence avec une exactitude parfaite.

Certes, ce processus est assez complexe, mais il vous donne un apperçu de ce que peut réaliser un tel programme. Il vous est possible, dorénavent, avec quand même une bonne dose de courage, de réaliser un processus qui pourrait modifier la valeur de tempo lorsque la séquence se déroule. Il faudrait pour cela incrémenter, ou décrémenter les valeurs de la table **time.t** suivant une courbe qui pourrait être aussi décrite dans une table de transfert. Le choix de la valeur à ajouter ou retrancher peut être fait en fonction de l'indice de *vélocité* que vous donnerez à la partition que vous jouerez et qui contrôlera le déroulement temporel de cette séquence. N'oubliez pas qu'une des grandes caractéristique de cette démarche est le contrôle en temps réèl de partitions virtuelles pouvant être modifiées suivant le jeu de l'interprête. C'est la version "moderne" des relations entre écriture et moyens techniques.

## 6.2. UNE SOLUTION PLUS GENERALE.

Soit la séquence suivante à réaliser. Les deux parties entre crochets sont des voix indépendantes, qui produisent des accords, pour la partie supérieure, et arpègent différemment ces accords dans la partie inférieure.




Quelle est le premier réflexe qui vient à l'esprit, en ce qui concerne les séquences arpégées ? Ce doit être celui de créer 4 tables supplémentaires pour chaque composant. Cela pose plusieurs types de problèmes. D'abord, il faudra changer le sequencer pour lui ajouter 12 tables (3 pour chacune des séquences); cela ferait 15 tables en tout. Ce n'est pas commode, et les risques d'erreurs sont assez grands. D'autre part, si, dans le cours de votre morceau, une solution analogue se présente, avec un nombre de séquences encore supérieur, votre patch sera de nouveau inadéquat. L'idée maîtresse est toujours de créer des patches qui soient les plus généraux possible dans un contexte donné, afin de pouvoir faire face à toutes les situations voisines quelles que soient la complexité de la structure. Une nouvelle solution s'impose donc. Elle consiste à inscrire le contenu de vos tables directement dans votre partition. Le procédé sera le suivant :

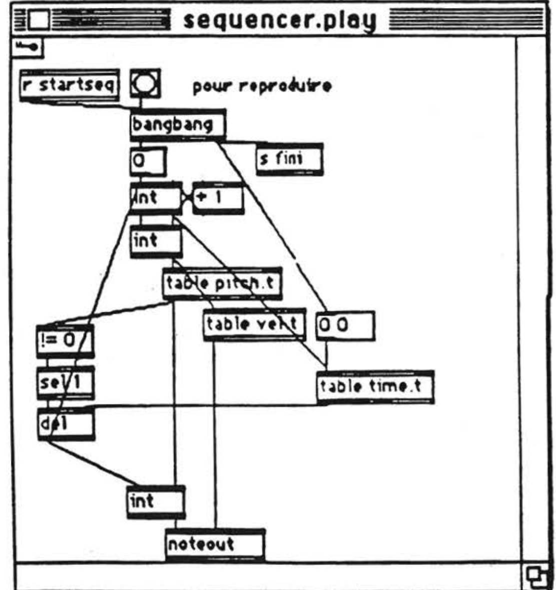
a) Vous jouez la première séquence (comme nous l'avons vu)
b) Vous faites figurer quelque part le contenu de vos tables dans des boîtes **messages**
c) Vous copiez ces boîtes dans le corps de votre partition (c'est une opération assez délicate)
d) Vous envoyez un message à la sortie des boîtes de votre partition au synthétiseur.

Pour réaliser cela, vous disposez de deux patches, **sequencer.write** qui vous permettra de faire figurer les valeurs des tables dans une boîte **message** que vous copierez ensuite dans votre partition, et **sequencer.play** qui se chargera de l'exécution des séquences. Pour plus de sûreté, vous pouvez faire figurer ces deux patches dans le corps de votre patch principal sous forme de **patcher**.

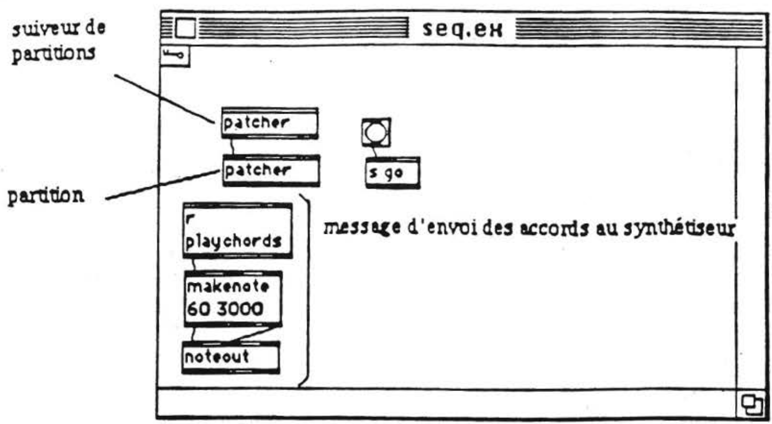
Chargez ces deux fichiers :

**Sequencer.write** est à peu près identique à **sequencer** mais avec les ordres d'écriture en plus. Les messages **append $1** recueillent le contenu des tables et l'écrivent dans les boîtes **message** du bas. Les messages **set \; <table>.t set 0** permettent d'inscrire en tête de ces boites les arguments en question. Vous les voyez d'ailleurs figurer à l'intérieur. Testez ce module dès maintenant. *Cliquez* sur le **bang** d'enregistrement puis sur celui d'écriture et vous verrez le contenu numérique de ce que vous avez joué apparaître dans les boîtes.

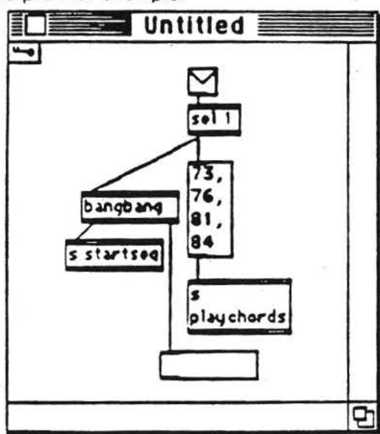
**Sequencer.play** contient la partie de reproduction de **sequencer** plus quelques messages qui vont lui permettre de communiquer avec **sequencer.write** et la partition.



Votre patch général doit se présenter comme suit, avec, c'est classique maintenant, le suiveur de partitions connecté à la partition, ainsi qu'un message **playchords** qui permettra d'envoyer les données des accords, figurant dans la partition, au synthétiseur :




La partition se trouve esquissée dans l'exemple suivant. Y figurent le premier événement dont n'est codé que l'accord. Vous constaterez que les accords transitent par un message **playchords** dont la fonction figure dans le patcher principal. D'autre part, les arpèges, dont le contenu figurera dans la boîte **message** vide pour l'instant, sont précédés de deux objets : **bangbang** qui permettra d'effectuer la lecture des valeurs *avant* le **s startseq** (n'oubliez pas que les sorties du **bangbang** s'effectuent successivement de droite à gauche). Le **r startseq** qui correspondra au **s startseq** se trouve dans **sequencer.play** et effectuera l'envoi des valeurs au synthétiseur comme nous l'avons vu dans le premier exemple.



Reste à voir comment on copie le contenu des boîtes de **sequencer.write** dans la partition. C'est ce que je vais vous montrer maintenant. Allez dans votre fenêtre **sequencer.write**, *cliquez* sur le bang et jouez le premier arpège :



*cliquez* ensuite sur le **bang** d'écriture et vous devez voir apparaître le contenu des trois tables (successivement **pitch.t**, **time.t** et **vel.t**) s'inscrire dans les trois boîtes messages comme c'est indiqué ci-dessous:

sequencer.write

pour écrire !

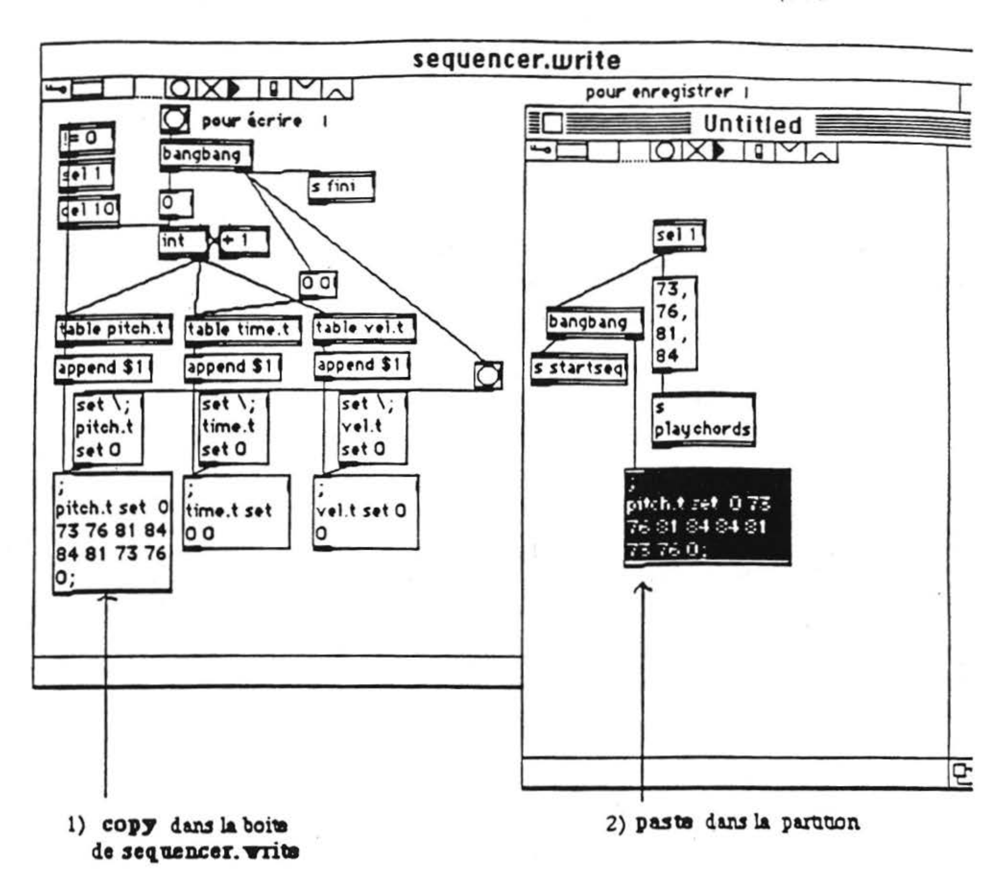
pour enregistrer !

!= 0
sel 1
del 10
bangbang
s fini
0
int
+ 1
0 0
table pitch.t
table time.t
table vel t
append $1
append $1
append $1
set \; pitch.t set 0
set \; time.t set 0
set \; vel.t set 0
; pitch.t set 0 73 76 81 84 84 81 73 76 0
; time.t set 0 0 90 104 93 2254 5 181 1 2169
; vel.t set 0 110 91
110 0 0 0 0 0
r fini
0
notein
int
bang
timer
0
int
+ 1
8
table time.t
table pitch.t
table vel.t

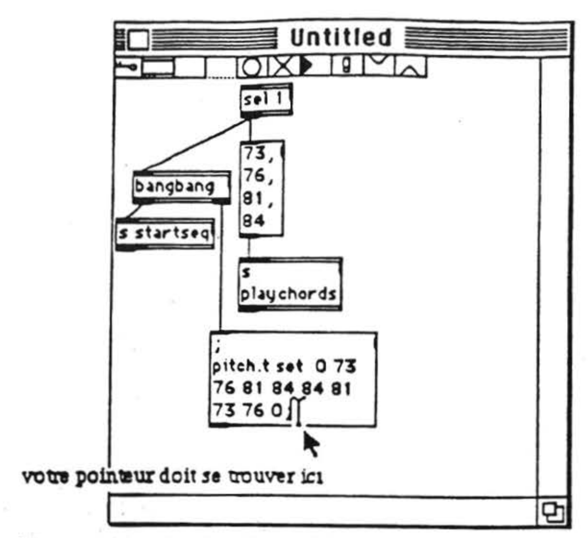
Regardons les une à une. La première table donne les valeurs 73 76 81 84 qui correspondent aux *Notes On* puis, 84 81 73 76 qui correspondent aux *Notes Off*. Cela indique, que ces *Notes On* été tenues un certain temps sur le clavier pour permettre une résonance. Avec un timbre qui aurait résonné naturellement sans tenues, il aurait été suffisant de lâcher les notes lorsqu'on jouait les suivantes. La table **time.t** donne les valeurs de départ, donc 0 90 104 et 93 pour les quatre *Notes On*, puis 2254 représente l'intervalle de temps entre la dernière *Note On* et la première *Note Off*, enfin 5, 181, 1 et 2169 indiquent les temps entre les différentes *Notes Off*. La table **vel.t**, quant à elle, donne les quatres vélocités des *Notes On* puis les *Notes Off* en remettant les vélocités à 0.

faux la dernière valeur le tps entre la dernière note off & l'appui du bouton "pour écrire"

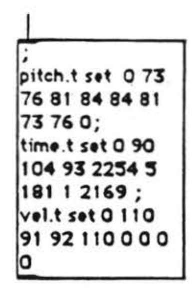
Inscrivons, maintenant, une à une, le contenu des tables dans la partition. Faites donc **copy** dans le sequencer, puis **paste** dans la partition comme c'est indiqué ci-dessous :




Le message figurant avant les valeurs numériques **pitch.t set**, **time.t set** et **vel.t set** provoqueront automatiquement l'écriture de ces valeurs dans les tables correspondantes. Ensuite ce sera le tour de la deuxième table. Répétez l'opération, mais positionnez vous au bon endroit dans la boîte message de votre partition . C'est à dire que vous devez positionner votre pointeur après le ";" qui termine la première série des valeurs de pitches que vous avez déjà inscrite, comme indiqué ci-dessous :




Lorsque vous ferez **paste** votre deuxième table ira s'inscrire à la suite de la première, et ainsi de suite pour la table des vélocités. A la fin, vous devez avoir toutes les valeurs comme ceci :




Ainsi, lorsque le programme aura lu les valeurs (à partir de la sortie droite du **bangbang**) il les enverra à **sequencer.play** par l'intermédiaire du **s startseq.**

**EXERCICE**

A partir de cet exemple, je vous incite maintenant à coder tout le reste de la partition, et de la tester. N'oubliez pas que si vous n'avez pas mis **sequencer.play** en tant que **patcher** dans votre programme principal, vous serez obligé de le *charger* chaque fois que vous désirerez faire tourner cet exemple.En cas de panique, vous avez toujours la solution de charger l'application **sequence** qui réalise cette esquisse.

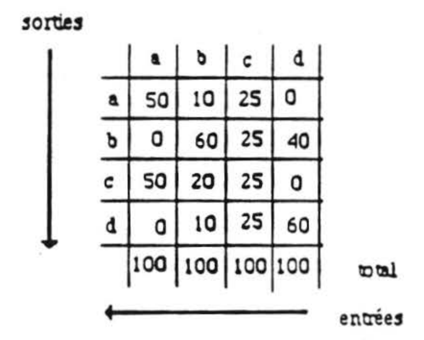
## 7. SUR QUELQUES CONFIGURATIONS NON-STANDARDS

Nous avons vu, jusqu'à présent ce que l'on peut appeler des *configurations standards*, c'est à dire des cas de figures utilisant des procédures assez classiques. Je vous propose maintenant d'étudier quelques configurations qui sortent de ce cadre, soit qu'elles ont été créées pour des besoins spécifiques, soit qu'elles font appel à des objets plus particuliers.

### 7.1. LES CHAINES DE MARKOV

Ce processus a déjà été utilisé en musique (Xénakis en particulier a produit plusieurs pièces à partir des chaînes de Markov). L'intérêt que l'on peut avoir en utilisant un tel processus doit dépendre de la manière dont on compte l'orienter. L'utilisation des lois probabilistes, en musique comme dans les autres arts, doit correspondre à un besoin précis. En effet, il est trop fréquent de voir des compositions basées sur ces principes sans que l'on en sente vraiment la nécessité artistique. Il est évidemment très facile de laisser à un processus le soin de choisir aléatoirement des valeurs lorsque l'imagination fait défaut. Aussi il est recommandable de bien savoir ce que l'on veut faire et dans quel but lorsqu'on se soumet à de telles règles. Mais avant d'entrer dans le vif du sujet, en voici les lignes générales.

Le but consiste à utiliser des matrices de probabilités qui assureront des transitions entre des éléments définis au préalable. Ces probabilités seront donc dirigées d'une manière assez rigoureuse. Imaginez un matrice de dimensions assez petite pour simplifier l'explication :




| | a | b | c | d |
|---|---|---|---|---|
| a | 50 | 10 | 25 | 0 |
| b | 0 | 60 | 25 | 40 |
| c | 50 | 20 | 25 | 0 |
| d | 0 | 10 | 25 | 60 |
| | 100 | 100 | 100 | 100 |

Comment l'interpréter ?

Vous avez quatre objets *a, b, c, d* qui doivent s'enchaîner entre eux. Vous entrez par les colonnes et vous sortez par les lignes comme c'est indiqué. Ce qui signifie que pour la première colonne l'objet *a* aura 50% de chance d'être suivi par lui-même, 0% par *b*, 50% par *c*, et 0% par *d*. A savoir que *a*, soit se répètera, soit ira sur *c*. Les autres chemins sont indiqués dans les colonnes correspondantes. Le total des colonnes doit être égal à 100. Ainsi vous voyez que *c* aura une équiprobabilité d'être suivi par les autres objets. Lorsqu'on tire des nombres au sort, on obtient une *chaine* d'objet qui dans ce cas pourrait donner : ***a a c d d b b c a a a c c b b d d*** etc.. Ce procédé est très simple dans son fonctionnement comme vous pouvez le voir.

Dans ma pièce "*Pluton*", je l' ai utilisé de la manière suivante. Imaginez qu'au lieu d'avoir une matrice de 4 entrées et 4 sorties, vous avez 88 colonnes et 88 lignes, soit un nombre correspondant aux touches de votre piano ou KX88 :

| | 1 | 2 | 3 | 4 | ....... | 87 | 88 |
|---|---|---|---|---|---|---|---|
| 1 | | | | | ....... | | |
| 2 | | | | | ....... | | |
| 3 | | | | | ....... | | |
| 4 | | | | | ....... | | |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ |
| 87 | | | | | ....... | | |
| 88 | | | | | ....... | | |

Maintenant, si vous jouez :



vous produisez la succession de pitches MIDI 60 , 66, 64, 71 ce qui vous donne dans la matrice les pointages suivants :

| | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
|---|---|---|---|---|---|---|---|---|
| 60 | | | | | | | | |
| 61 | | | | | | | | |
| 62 | | | | | | | | |
| 63 | | | | | | | | |
| 64 | | | | | | | ■ | |
| 65 | | | | | | | | |
| 66 | ■ | | | | | | | |
| 67 | | | | | | | | |
| 68 | | | | | | | | |
| 69 | | | | | | | | |
| 70 | | | | | | | | |
| 71 | | | | | ■ | | | |
| 72 | | | | | | | | |

Reste maintenant à définir les probabilités. Si on sait quelles sont les successions, on ne connait pas les probabilités. Celles-ci seront définies par l'indice de vélocité, suivant la règle énonçant que, *dans un couple d'événements, plus la seconde note sera forte, plus la probabilité de succession sera élevée.* Donc dans le cas :



nous pourrons avoir une matrice de type :

| | 60 | 61 | 62 | 63 | 64 | 65 | 66 |
|---|---|---|---|---|---|---|---|
| 60 | | | | | | | |
| 61 | | | | | | | |
| 62 | | | | | | | |
| 63 | | | | | | | |
| 64 | | | | | | | 30 |
| 65 | | | | | | | |
| 66 | 80 | | | | | | |
| 67 | | | | | | | |
| 68 | | | | | | | |
| 69 | | | | | | | |
| 70 | | | | | | | |
| 71 | | | 10 | | | | |
| 72 | | | | | | | |

Les durées, il faut le signaler, sont aussi prises en compte dans les successions des notes. C'est donc un couple hauteur-durée qui sera mémorisé dans la matrice. Quelles conclusions donner à tout cela ?
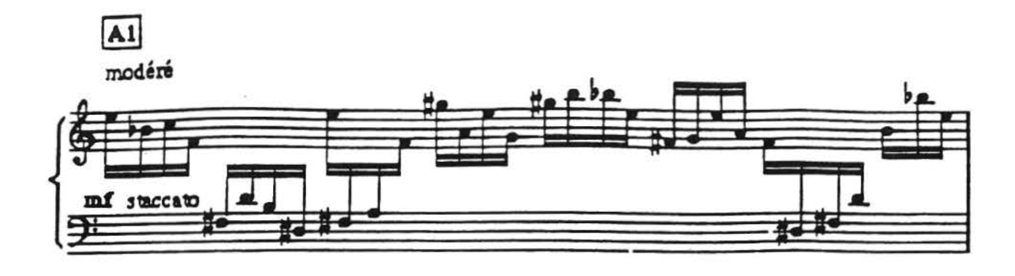
1) Etablir des séquences qui puissent s'enchaîner les unes aux autres par le biais de notes communes.
2) Permettre un discours qui reproduira "aléatoirement" les figures jouées par le pianiste. Et, de ce fait, créer une texture musicale qui sera en imitation par rapport au jeu du soliste.
3) Organiser une forme *en temps réel* qui se transformera petit à petit au fur et à mesure que le soliste ajoutera de nouvelles séquences.
4) Donner au principe de *forme ouverte*, une signification, non purement conceptuelle, mais bien perceptuelle. En effet, le choix "aléatoire" dans l'ordre de succession des séquences aura une répercussion immédiate dans la réponse de la machine.
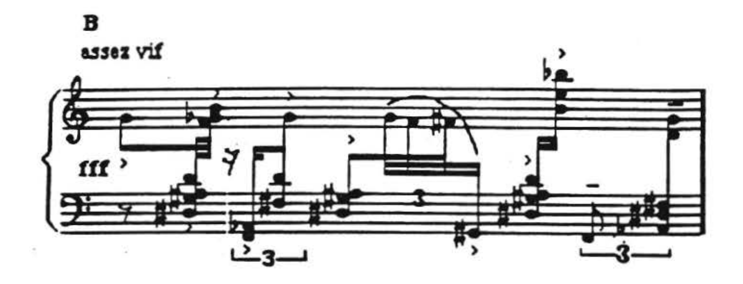
On voit, ainsi, qu'un principe, même simple comme celui-ci, dans son fonctionnement, peut engendrer tout un concept formel qui peut être riche en résultats. Nous allons tenter d'en faire l'expérience.
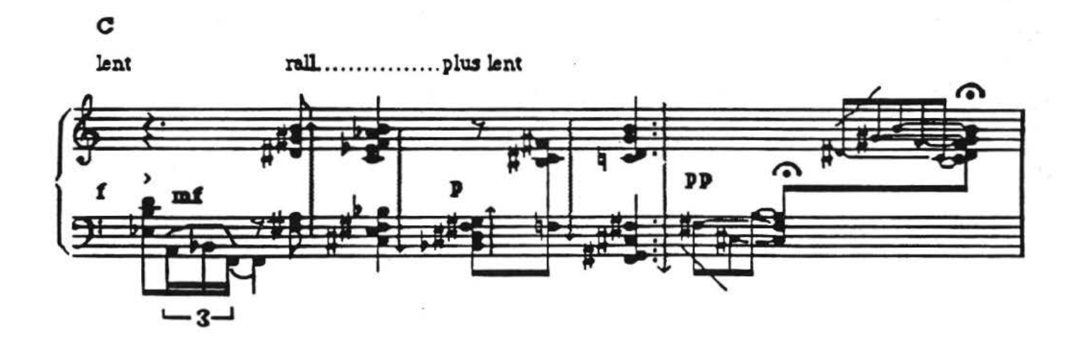
Voici une partition. Elle se compose de 6 séquences dont quatre sont principales, A1, B, C et D1, et deux sont dérivées, A2 et D2. L'écriture de cette partition suppose que :

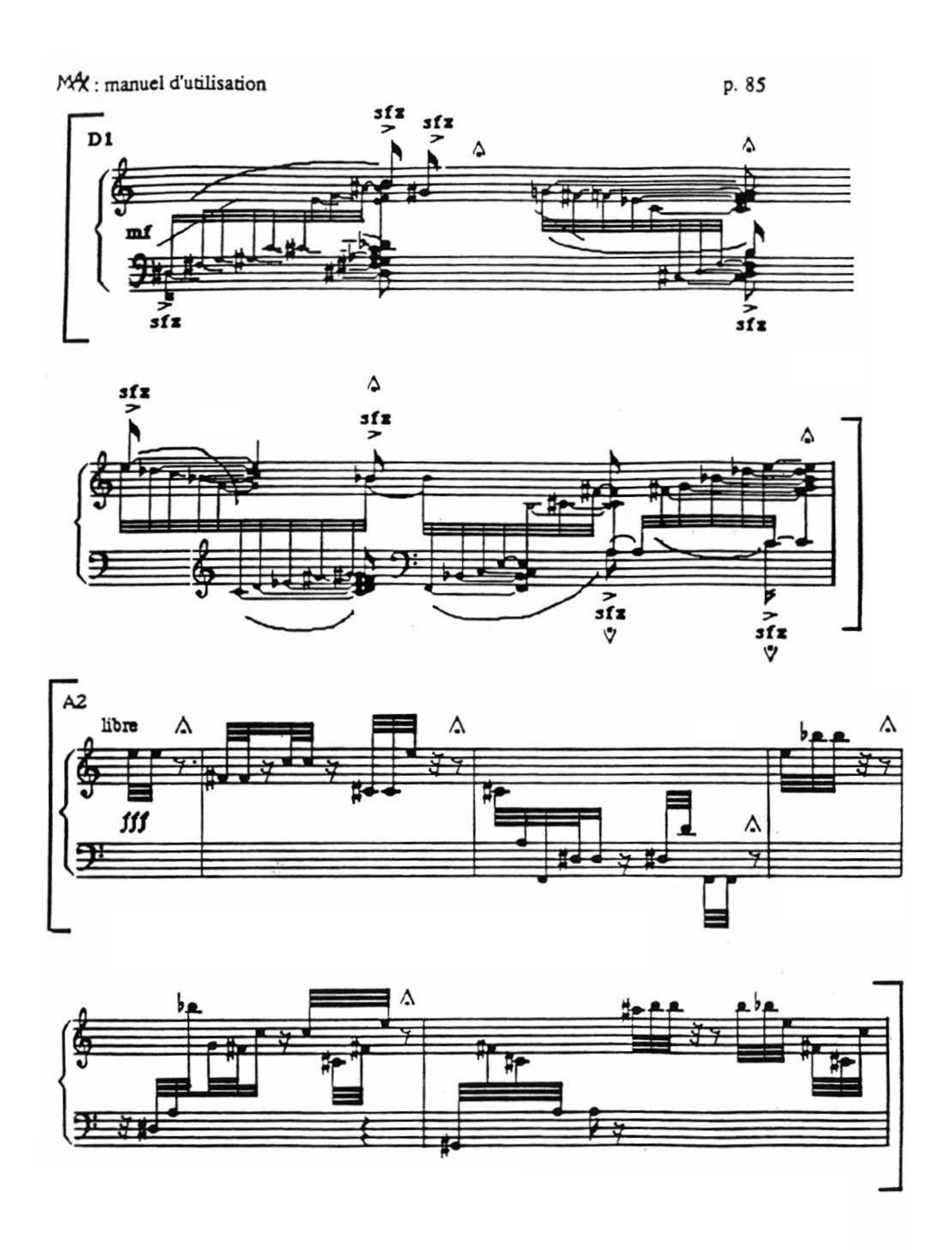
a) chaque séquence soit autonome, c'est à dire qu'elle peut se développer à partir d'elle-même par le seul fait qu'elle comporte des notes identiques se reproduisant dans le même registre, et par là, peut *court-circuiter* les trajets musicaux,

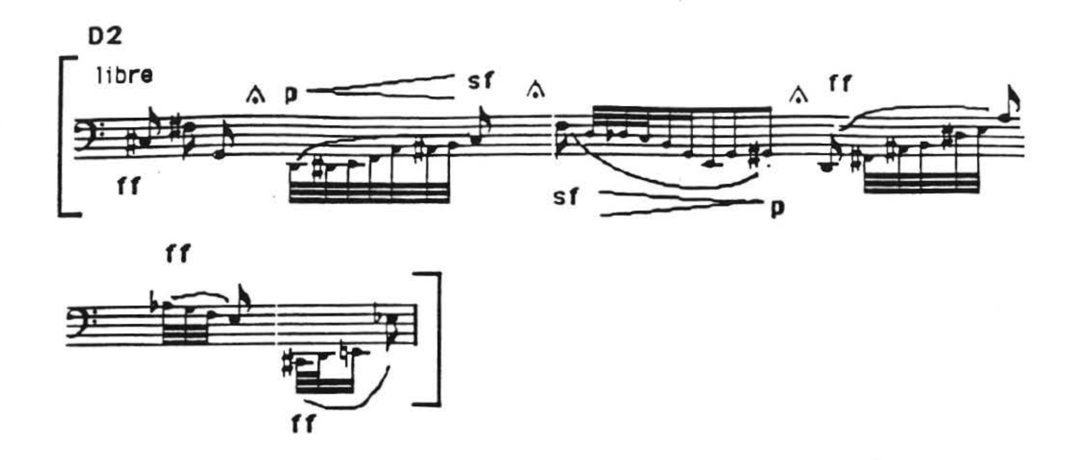
b) chaque séquence puisse s'enchaîner avec n'importe laquelle des cinq autres par le biais des notes communes.


A1
modéré
mf staccato


B
assez vif
fff
3
3
3


C
lent
rall................plus lent
f
mf
p
pp
3


D1
sfz
sfz
sfz
mf
sfz
sfz
sfz
sfz
sfz
sfz
A2
libre
fff




L'ordre des séquences est théoriquement complètement au choix de l'interprète. Cependant, pour des raisons musicales, donc non rigoureusement théoriques, les ordres de successions valables demeurent, à mon sens, les suivants:

**A1 A2 C B D1 D2** ou **C B A1 A2 D1 D2**

Chargez l'application **markov**. Vous avez un message **record** dans lequel vous allez *cliquer* avant d'exécuter une séquence. Je vous propose l'expérience suivante.

1) Cliquez donc dans **record**.

2) Jouez **A1**, la séquence se prolifère en elle-même dans votre synthétiseur.

3) Entrez peu à peu les événements de **A2** pour pouvoir apprécier " l'effet d'incrustation " d'une séquence dans l'autre. Vous constatez que le principal agent de distorsion structurelle est la polarisation sur les notes répétées qui figent par moments **A1**.

4) Jouez **C** en prenant soin d'arpéger lentement les accords et de bien faire sentir les modifications de tempi inscrites dans la partition. Vous constaterez que les arpèges s'enchaînent entre eux parfois, et que la rigidité de tempo de **A1** et **A2** est alanguie par la flexibilité de **C**.

5) Jouez **B**. Une nouvelle structuration apparait, celle des brusques accords **ff**. Vous constatez aussi que **A1** disparait, peu à peu laissant place aux nouvelles séquences.

6) Jouez **D1** puis **D2** pour achever le processus.

Testez ensuite chaque séquence séparément pour bien entendre les variations produites. N'oubliez pas que *tout est affaire d'interprétation* dans ce contexte, et que vous devez tester vos interprétations en fonction des types de résultats que vous voulez obtenir.

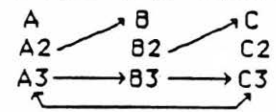
Vous avez, avec ce mode d'intégration entre instrument et système en temps-réel, un exemple type de ce que j'appelle les *partitions virtuelles* dont je donne ici la définition: partition dont on connait a priori la nature des composants à traiter (hauteurs, rythmes, intensités ...) mais dont on ignore les valeurs exactes affectées à ces composants avant qu'elle soit exécutée. Ce mode de communication entre le musicien et un système ouvert me semble être parmi les plus prometteurs dans le contexte de la musique électronique vivante.

**EXERCICE.**

Ce patch vous donne une matière assez riche à l'expérimentation, et je vous incite soit à compléter cette partition en introduisant d'autres séquences de manière à obtenir une structure entièrement prolifique, soit à composer une partition entièrement nouvelle en respectant les règles d'enchaînements qui la font fonctionner. Je vous indique quelques directions supplémentaires:

Soit trois séquences A, B et C ne communiquant pas entre elles, c'est à dire pour le système, n'ayant aucune note en commun.

Composez une série de séquences dérivées, A2, A3, B2, B3, C2, C3 dont vous établirez les réseaux de la manière suivante (les flèches indiquent la possibilité d'enchaînement) :



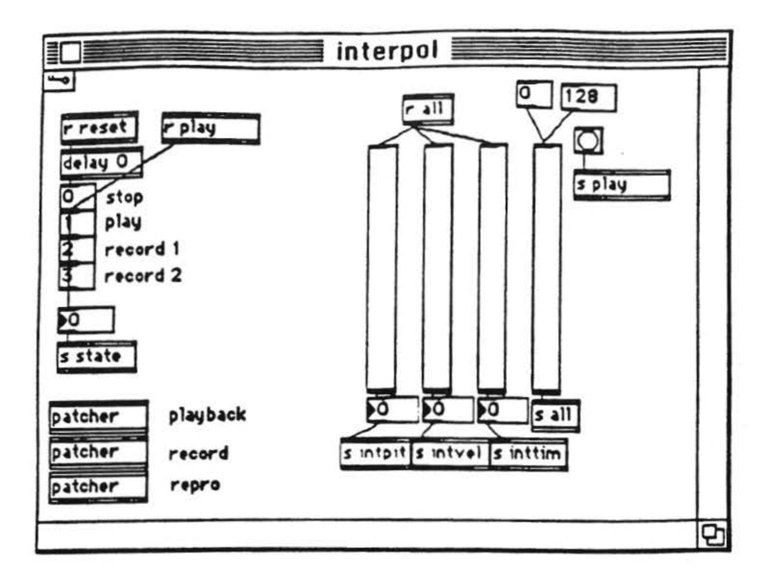

N'oubliez pas que les indices de vélocité déterminent les pourcentages de succession, et que vous pouvez jouer sur différents degrés de relations suivant le nombre de notes communes aux séquences devant s'enchaîner.

D'autre part, pour être bien perceptible, ce processus devra être composé suivant différents modes de structuration. A, B et C devront mettre en jeu des catégories temporelles très distinctes (tempi fixes, variables, continus, discontinus etc...). A partir de cela vous pouvez composer vos séquences dérivées de manière à obtenir en A3, B3 et C3 une relative indifférenciation structurelle pour pouvoir obtenir des états différenciés, indifférenciés et intermédiaires.

## 7.2. INTERPOLATIONS DE MELODIES, RYTHMES ET INTENSITES

Ce patch est assurément le plus complexe que nous ayons à voir dans ce manuel, c'est pourquoi je l'ai réservé pour la fin. Le principe, assez puissant du point de vue musical, est de réaliser des interpolations entre deux fragments musicaux, mais de manière à pouvoir interpoler les hauteurs, les vélocités et les rythmes de manière indépendante.
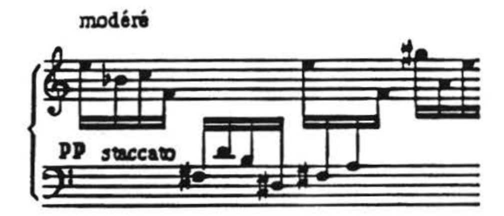
Chargez l'application **interpol**.



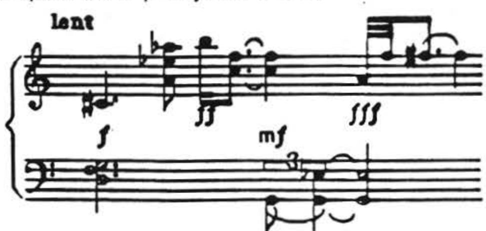

Vous avez quatre **messages** effectuant différentes fonctions (2 et 3 : enregistrements des deux fragments à interpoler, 1 : reproduction et 0 : stop). Les trois premiers **sliders** contrôlent, successivement de gauche à droite, les interpolations de pitches, vélocité et durées, le quatrième permet, par l'intermédiaire de la transmission du message **all**, de modifier en parallèle les trois précédents.

Faites l'expérience suivante.

1) *Cliquez* sur **2** puis jouez ce fragment (en respectant la nuance indiquée)




2) *Cliquez* sur **3** puis jouez celui-ci




3) Positionnez le **slider** général à 0 et *cliquez* sur **1** (play), vous entendrez la première séquence, positionnez le maintenant au maximum, la seconde apparaîtra.

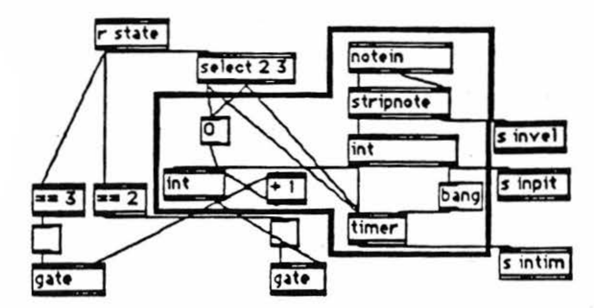
3) Positionnez le **slider** général à 0 et *cliquez* sur **1** (play), vous entendrez la première séquence, positionnez le maintenant au maximum, la seconde apparaîtra.

4) Si vous inversez les **sliders** de pitches, d'une part, et de vélocité et de rythmes d'autre part, vous aurez le matériel mélodique ou harmonique d'une séquence avec les valeurs de durées et de vélocités de l'autre. Dans l'exemple ci dessous, c'est assez clair, car la première séquence étant monodique, vous calquerez cette configuration rythmique sur les accords de la seconde, et vice et versa.

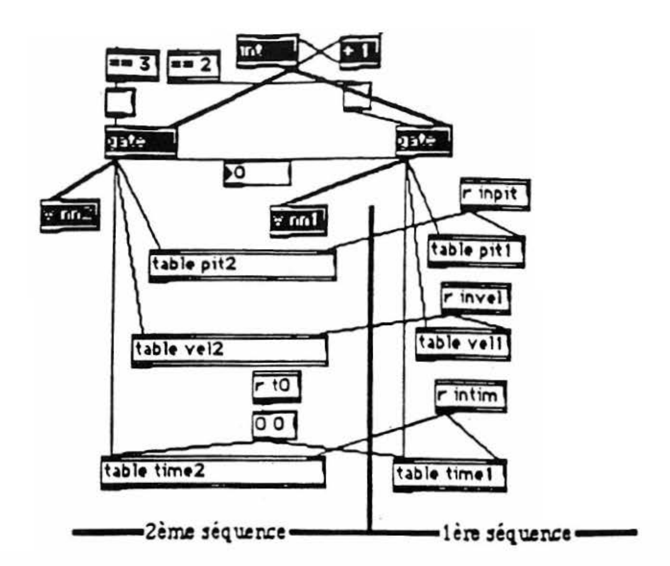
Vous pouvez aussi interpoler les mélodies mais vous risquerez d'obtenir des rencontres qui échapperaient à votre contrôle car, et c'est mon sentiment personnel, les hauteurs, étant les composants où le pouvoir discriminatoire de la perception est le plus fort, ne peuvent pas se manipuler impunément au même titre que les durées et les intensités, faisant référence, eux, à un mode de perception plus flou et plus global. Si vous désirez interpoler les pitches vous aurez tout intérêt à choisir un timbre inharmonique ou la sensation de hauteur est plus complexe. Vous éviterez ainsi des rencontres mélodiques ou harmoniques fâcheuses.

Si le maniement de ce patch est des plus simple, sa construction est très complexe, et je vous propose d'en regarder les détails.

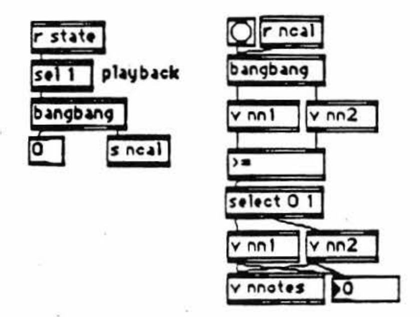
Vous avez sans doute remarqué que ce patch en contient d'autres. *Cliquez* dans le **patcher** "record" pour faire apparaître le deuxième niveau. :



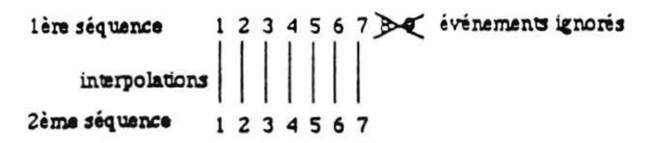

Dans cet extrait, la partie encadrée est déjà connue. Nous l'avons détaillée dans l'explication concernant le **sequencer**, il s'agit de l'arrivée des valeurs MIDI, et du compte des événements dans une **boucle** (cf paragraphe **6.1**). La sortie de ces valeurs se transmettra par des messages (**invel** pour les vélocités, **inpit** pour les pitches et **intim** pour les temps) qui iront s'inscrire dans les tables correspondantes. L'utilisation de ces transmissions de messages se fait ici dans un but de clarification graphique, mais il est possible aussi de connecter directement ces sorties dans les tables. Ce processus sera déclenché lorsque le message **state** recevra les valeurs **2** ou **3** (correspondantes aux fonction **record 1** et **record 2** du patch **interpol**. Lorsque la valeur sera **2** le **gate** de droite s'ouvrira pour remplir les tables **pit1**, **vel1** et **time1**, c'est à dire les tables mémorisant les valeurs de la première séquence, lorsqu'elle sera **3**, les trois autres tables se rempliront lors de la deuxième séquence, comme le montre la construction suivante :



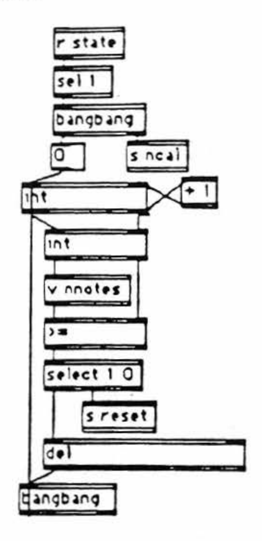

Les entrées des **gates** recevront l'incrémentation des événements joués sur le clavier et transmettront, par leurs sorties, les nombres d'événements de chaque séquence aux objets **v nn1** et **v nn2.** L'objet **v** ou **value** est voisin de **send** à ceci près qu'il ne transmet les valeurs que lorsqu'on lui demande et non sans arrêt. Dans ce cas **value** recevra successivement 0, 1, 2 etc.. jusqu'à la valeur finale, mais il ne transmettra la valeur que dans le cas suivant. Entrez dans le **patcher** "playback" dont voici deux extraits :



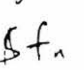

Lorsque vous **enverrez 1** à la transmission du message **state**, un **bangbang** déclenchera un **bang** dans la transmission **ncal**. Celle-ci testera si **nn1** *est plus grand ou égal à* **nn2**. Si "oui", le **comparateur** enverra **1**, et le **select** choisira **nn2**, et vice et versa. Ceci pour choisir, **dans tous les cas**, le plus petit nombre d'événements pour l'interpolation et ignorer ceux qui **restent** :



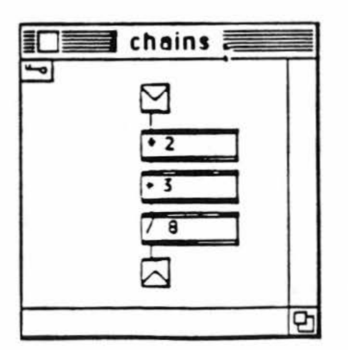
La valeur sélectionnée sera transmise à **v nnotes** dans l'extrait suivant qui comptera les événements pour l'interpolation :

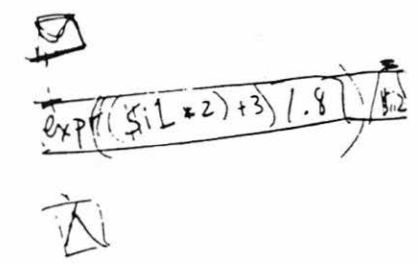


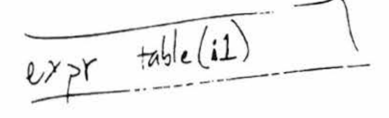

Il y a deux objets **int** connectés entre eux. Le premier fera une incrémentation à chaque **bang** (issus de l'objet **bangbang**), et le second transmettra la valeur incrémentée au comparateur **>=** qui testera si **v nnotes** est plus grand ou égal à la valeur courante, c'est à dire si, la dernière note du fragment le plus court étant arrivée, on doit stopper l'interpolation. Dans ce cas le **select 1 0** se dirigera vers un **bangbang** nous ramenant au début de ce processus pour continuer, ou bien produira un **bang** dans **s reset** pour arrêter (cf cette transmission de message dans le patch **Interpol**). Reste à voir les processus d'interpolations proprement dits.



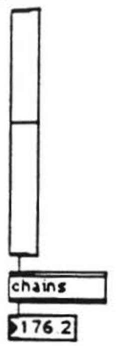
Toujours dans le même patch ("playback"), vous remarquez trois objets **intertab** qui sont finalement des **patchers** assez particuliers. Un petit exemple va vous en montrer l'utilité. Imaginez que vous avez à effectuer très souvent une même opération (disons une chaîne d'opérations). Pour cela vous allez créer un nouveau **patcher** qu'on appelera **chains** :



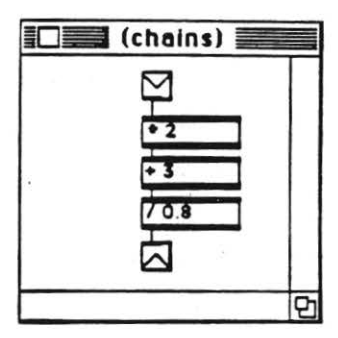


La chaîne d'opération est précédée d'un **inlet** et suivie par un **outlet**. Maintenant dans une autre fenêtre vous allez appeler ce processus de la manière suivante :



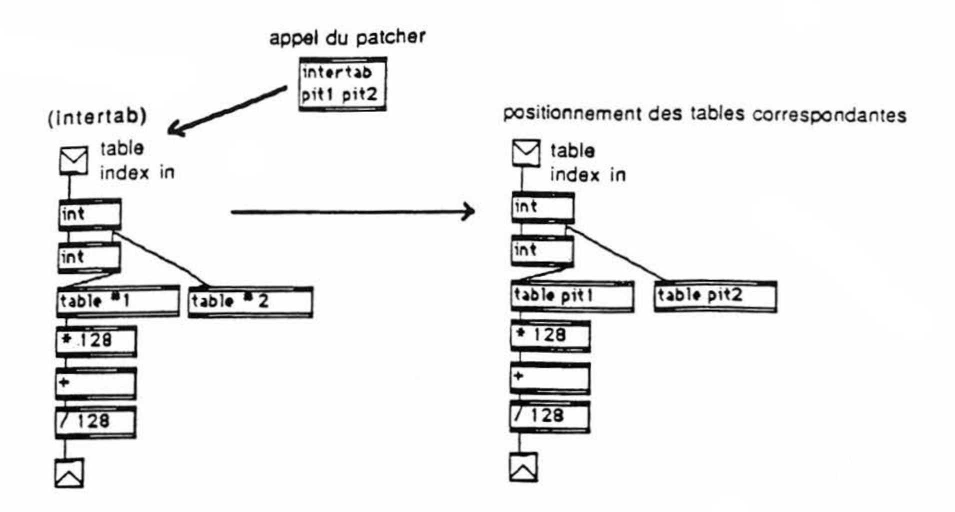

et vos opérations seront effectuées. Ceci est utile surtout dans le cas d'une série d'opérations plus compliquées que dans celle-ci.

(Regardez un trait distinctif. Si vous voulez ouvrir le patcher **chains** en *cliquant* dessus, il apparaîtra de la manière suivante :

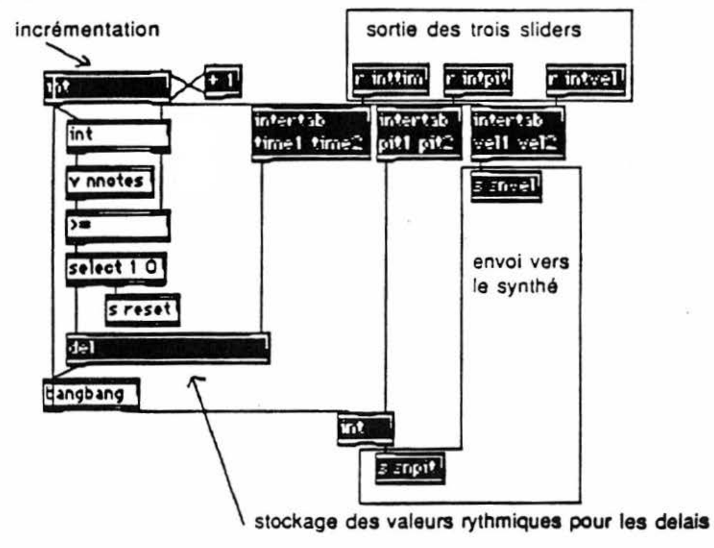



Son nom est entre parenthèse, et vous ne pouvez plus l'éditer, les symboles comme la clé ne sont plus disponibles).

C'est exactement ce qui se passe pour effectuer les interpolations dans notre exemple. Les patchers **intertab pit1 pit2 ... tim1 tim2** etc... font appel au patcher **intertab** qui remplacera les arguments **#1 #2** par ceux indiqués dans la boite d'appel comme le montre l'exemple suivant :




L'algorithme d'interpolation serait trop fastidieux à expliquer. Je vous propose de le tester en mettant des **numbers** à la sortie de chaque opérateur. Disons, pour simplifier qu'il prend en entrée les deux valeurs correspondantes dans chaque table (celle de la première et de la seconde séquence). Suivant le position de votre **slider** il choisira une valeur intermédiaire. Plus la valeur du **slider** sera élevée plus la valeur interpolée sera proche de la deuxième séquence et vice et versa. Les trois **sliders** enverront des messages **inttim, intpit, intvel,** qui entreront dans les **inlets** gauches de **intertab**, les sorties de ce module se dirigeront vers des messages (**snpit** et **snvel**) envoyés au patcher réservé à l'entrée du synthétiseur ("repro "). Les valeurs rythmiques seront stockées dans le **del** qui prendra le temps issu de l'interpolation avant de continuer l'incrémentation des valeurs dans les tables :



# 8. LEXIQUE ALPHABETIQUE DES PRINCIPAUX OBJETS.

N'oubliez pas que vous pouvez consulter la structure des objets à n'importe quel moment. La liste suivante vous indique simplement leurs noms et leurs fonctions mais cliquez sur la touche "option" (marquée du symbole ⌥) et *cliquez* sur l'objet en question pour avoir toutes les informations nécessaires (en anglais).

## 8.1. MIDI

**bendin** reçoit une valeur de pitch-bend du synthétiseur et la transmet
**bendout** envoie une valeur de pitch-bend au synthétiseur
**ctlin** reçoit une valeur de contrôle MIDI du synthétiseur et la transmet
**ctlout** envoie une valeur de contrôle MIDI au synthétiseur
**makenote** crée un *note-off*
**midiformat** reçoit des valeurs MIDI et les renvoie octet par octet
**midiin** reçoit les octets correspondants au message MIDI
**midiout** renvoie les octets correspondants au message MIDI
**midiparse** reçoit des valeurs octet par octet et renvoie une valeur MIDI
**notein** reçoit les valeurs *canal-vélocité-pitch* du synthétiseur et les transmet
**noteout** envoie les valeurs *canal-vélocité-pitch* au synthétiseur
**stripnote** supprime le *note-off*
**touchin** reçoit une valeur d'*after-touch* du synthétiseur et la transmet
**touchout** envoie une valeur d'*after-touch* au synthétiseur

## 8.2. MESSAGES

**bangbang** reçoit un bang et renvoie deux bangs l'un après l'autre
**int** reçoit puis envoie un entier et produit un bang
**float** reçoit puis envoie un flottant et produit un bang
**pack** reçoit un plusieurs nombres et renvoie une liste
**swap** reçoit deux nombres et les inverse
**unpack** reçoit une liste et renvoie plusieurs nombres (inverse de **pack**)
**fswap** reçoit deux flottants et les inverse (si l'on donne un argument flottant)

## 8.3. TEMPS

**del** renvoie un valeur de retard en milliseconde
**line** produit une évolution continue entre deux valeurs
**metro** renvoie une série de bangs réguliers en millisecondes (métronome)
**pipe** très amusant. Mémorise une évolution et la reproduit après un retard
**speedlim** sélectionne certaines valeurs (suivant un tempo défini) dans une évolution continue
**timer** calcule la valeur (en millisecondes) entre deux bangs

## 8.4. OPERATEURS ARITHMETIQUES (cf. chapitre 2.5)

[% 25] modulo

[+] [-] [*] [/]

## 8.5. OPERATEURS LOGIQUES (cf. chapitre 2.6)




## 8.6. AUTRES

**print** envoie un ordre d'impression
**receive** reçoit une valeur d'un **send** (en continu)
**sel** sélectionne des valeurs numériques
**send** envoie une valeur à un **receive** (en continu)
**value** identique à **sel** + **receive** mais transmet une valeur lorsqu'on lui demande

# 9. LISTE DES APPLICATIONS, PARTITIONS ET TABLES DU MANUEL

Cette liste vous donne le nom de tous les éléments que vous devez posséder pour exécuter les exemples de ce manuel. Il doit y avoir 32 éléments en tout, dont 17 applications, 5 partitions et 10 tables auxquels il faut ajouter le programme MAX lui-même. Pour information, les exemples de ce manuel ont été testés avec la version MAX 1.15.13.

## 9.1. APPLICATIONS

(chains)
delslider
interpol
local
logic
multipatchers
multisliders
opera
partition
partition2
sequence
sequencer
sequencer.play
sequencer.write
sliderline
slidertable
suiveur

## 9.2. PARTITIONS

local.sc
part1.sc
part2.sc
part3.sc
seq.sc

## 9.3. TABLES

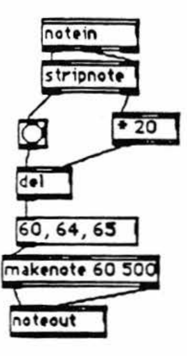
local.t
part1.t
part2.t
part3.t
seq.t
t1.t
t2.t
t3.t
t4.t
t5.t

# 10. QUELQUES PRECISIONS SUPPLEMENTAIRES.

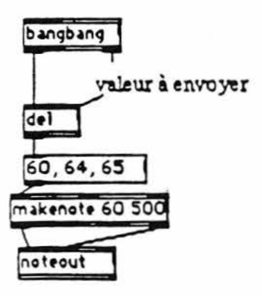
## 10.1. LES ENTREES ET SORTIES NON-SYNCHRONES.

De nombreux problèmes peuvent se poser si l'on ne tient pas compte du fait que certains objets qui transmettent les sorties ou recoivent les entrées dans un ordre qui doit être déterminé et non simultanément. Voyons quelques cas.

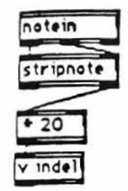
a) Voici un cas simple. Vous désirez envoyer une action avec un certain retard. Ce retard peut être contrôlé par l'indice de velocité suivant la relation établissant que plus la valeur de vélocité est faible, plus le retard sera bref. Voici d'abord ce qu'il ne faut pas faire. Le **notein** par l'intermédiaire du **stripnote** recueille les indices **de velocité** non-nuls. Cette valeur, passant par un coefficient multiplicatif est envoyée au **del** en même temps que celui-ci est actionné (par le **bang**). Il se produira des confusions car le **del** doit recevoir sa valeur avant d'être actionné :



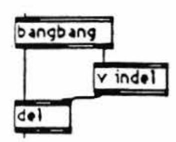

Il faut donc que cette valeur soit envoyée *avant* le moment où il recevra un **bang.** Vous savez qu'il existe un objet qui permet d'ordonner ces ordres d'envoi: **bangbang**. Dans l'exemple qui suit, la valeur en question sera envoyée en premier car **bangbang** produit deux **bangs** dont le premier est celui de droite. Ensuite celui de gauche enverra l'ordre d'action une fois la valeur positionnée dans le **del** :



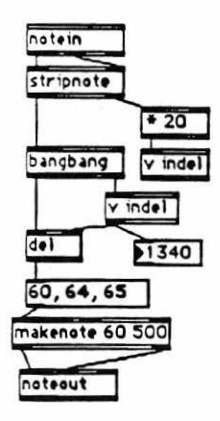

Comment faire pour recueillir la valeur de vélocité et la transmettre au **bangbang** ? Il faut la mémoriser dans un objet et la recueillir ensuite. L'objet idéal pour une telle procédure est **value** qui transmettra la valeur au moment où on lui demandera, c'est à dire à la sortie du **bangbang.**




La valeur mémorisée sera ensuite récupérée avant son envoi au **del** :




Il faut, pour terminer, prendre bien soin de la manière dont on va actionner le **bangbang**. Si l'action provient du **notein** vous aurez deux retards, un pour l'attaque (**Note On**), l'autre pour le relâchement de la touche (**Note Off**). C'est donc à partir du **stripnote** que devra être fait le déclanchement comme le montre le patch complet ci-dessous:




**En résumé on peut énoncer que dans tout objet possédant plusieurs entrées, on doit s'assurer que celle de gauche doit parvenir en dernier, l'ordre des autres étant sans conséquences.**

# APPENDICE

# APPENDICE

## TABLEAU DES HAUTEURS MIDI

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Oct -2* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| *Oct -1* | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| *Oct 0* | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| *Oct 1* | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| *Oct 2* | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| *Oct 3* | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| *Oct 4* | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| *Oct 5* | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| *Oct 6* | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 |
| *Oct 7* | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |
| *Oct 8* | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | | | | |

TABLE DES MATIERES

1. GENERALITES ........ 1

1.1. En quoi consiste MAX ? ........ 1
1.2 Le standard MIDI........ 2
1.2.1. Introduction ........ 2
1.2.2. Un réseau de communication musicale ........ 2
1.2.3. Structure des signaux MIDI........ 3
1.2.4. La transmission des données MIDI ........ 4
1.2.5. Les messages Canal........ 5
1.2.6. Les messages Voix ........ 5
1.2.7. Les messages Mode ........ 6
1.2.8. La fonction de contrôle local ........ 7
1.2.9. Les messages Système........ 7
1.2.10. Les fichiers MIDI........ 7

2. POUR DEBUTER EN MAX SANS SYNTHETISEUR........ 9

2.1. LE MENU 9
2.2. LES TABLES DE TRANSFERT ........ 16
2.3. LES OBJETS TEMPORELS........ 22
2.3.1 del (del x). ........ 22
2.3.2 line (line x y)........ 22
2.3.3 timer. ........ 23
2.4. LES TRANSMISSIONS DE MESSAGES ........ 23
2.5. LES OPERATEURS ARITHMETIQUES ........ 25
2.6. LES OPERATEURS LOGIQUES ........ 26

3. POUR DEBUTER EN MAX AVEC SYNTHETISEUR........ 27

3.1. LES SPECIFICATIONS MIDI........ 27
3.1.2. midiout. ........ 28
3.1.3. notein (ou notein n) ........ 28
3.1.4. noteout (ou noteout n) ........ 28
3.1.5. stripnote........ 29
3.1.6. makenote (ou makenote m ou makenote m n)........ 30
3.2. EXEMPLE COMMENTE........ 31

4. LE SUIVEUR DE PARTITIONS........ 37

4.1. UN EXEMPLE DE FONCTIONNEMENT........ 37
4.2. UN CAS D'UTILISATION SIMPLE ........ 38
4.3. UNE SOLUTION PLUS GENERALE........ 40
4.3.1. L'ENREGISTREMENT DE LA PARTITION ........ 41
4.3.2. LA SYNCHRONISATION DE LA PARTITION AVEC LES ACTIONS. ........ 43
4.4. UN EXEMPLE DE REALISATION........ 46
4.5. LA SUCCESSION DES PHRASES. ........ 50
4.6. LE RACCORDEMENT DES SECTIONS........ 51
4.7. LE PROBLEME DES ERREURS D'EXECUTION. ........ 55
4.8. LE PROBLEME DES ACCORDS........ 58

4.9. DES CONFIGURATIONS PROBLEMATIQUES. 59
4.9.1. Les notes répétées. 59
4.9.2. Les événements en nombre indéterminé 60
4.9.3. les passages de grande virtuosité 61

5. LES ACTIONS LOCALES 65

6. LE CODAGE DES SEQUENCES 71
6.1. UN EXEMPLE DE SEQUENCER 71
6.2. UNE SOLUTION PLUS GENERALE 74

7. SUR QUELQUES CONFIGURATIONS NON-STANDARDS 81

7.1. LES CHAINES DE MARKOV 81
7.2. INTERPOLATIONS DE MELODIES, RYTHMES ET INTENSITES 87

8. LEXIQUE ALPHABETIQUE DES PRINCIPAUX OBJETS 95

8.1. MIDI 95
8.2. MESSAGES 95
8.3. TEMPS 95
8.4. OPERATEURS ARITHMETIQUES (cf. chapitre 2.5) 95
8.5. OPERATEURS LOGIQUES (cf. chapitre 2.6) 96
8.6. AUTRES 96

9. LISTE DES APPLICATIONS, PARTITIONS ET TABLES DU MANUEL 97

9.1. APPLICATIONS 97
9.2. PARTITIONS 97
9.3. TABLES 97

10. QUELQUES PRECISIONS SUPPLEMENTAIRES 99

10.1. LES ENTREES ET SORTIES NON-SYNCHRONES 99


APPENDICE

TABLEAU DES HAUTEURS MIDI